Metabolic Network Data Integration and Visualization Software

Von der Fakultät für Lebenswissenschaften

der Technischen Universität Carolo-Wilhelmina

zu Braunschweig

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

genehmigte

D i s s e r t a t i o n

von Timo Lühr

aus Cuxhaven

1. Referent:	Prof. Dr. Dietmar Schomburg
2. Referent:	Prof. Dr. Dieter Jahn
eingereicht am:	11.12.2013
mündliche Prüfung (Disputation) am:	26.02.2014

Druckjahr 2014

Vorveröffentlichungen der Dissertation

Teilergebnisse aus dieser Arbeit wurden mit Genehmigung der Fakultät für Lebenswissenschaften, vertreten durch den Mentor der Arbeit, in folgenden Beiträgen vorab veröffentlicht:

Tagungsbeiträge

Luehr, T. & Schomburg, D.: MetaboliteExplorer (2009): A GUI based program for the managment of metabolome data, their visualization and database assisted storage of metabolite libraries (Poster) German Conference of Bioinformatics

Scheer, M., Quester, S., Luehr, T., & Schomburg, D. (2010): BRENDA Map of Biochemical Reactions and Metabolic Pathways (Poster) German Conference of Bioinformatics

Luehr, T., Scheer, M., Busch, M, Rother, M., Schomburg, I. & Schomburg, D. (2011): BRIME: A tool for visualization of metabolic models (Poster) German Conference of Bioinformatics

Für meine Großeltern

Danksagung

Diesen Abschnitt möchte ich nutzen, um nachfolgenden Personen zu danken. Ohne sie hätte ich diese Arbeit nie vollenden können.

Zuerst möchte ich Herrn Prof. Dr. Dietmar Schomburg danken, der mir die Möglichkeit gab an seinem Institut meine Doktorarbeit durchzuführen. Ich habe hier ein höchst interessantes Thema bearbeiten können.

Herrn Prof. Dr. Dieter Jahn möchte ich herzlich für die Übernahme des Zweitgutachtens danken.

An zweiter Stelle muss und soll meine Familie stehen, die mich immer unterstütze und immer an meiner Seite stand. Hier gilt mein ganzer besonderer Dank meinen Großeltern aus Cuxhaven. Habe Euch sehr lieb! Und meiner Oma Doris, vermisse Dich sehr! Euch widme ich diese Arbeit!

Natürlich geht mein Dank auch an all meine Freunde, die für mich da waren und Verständnis zeigten, dass die Freizeit innerhalb einer Dissertation mehr als knapp bemessen ist. Hier zähle ich niemanden auf, denn Ihr seid alle gemeint!

Aber da waren auch viele wichtige Arbeitskollegen im Institut, von denen ich manche nun zu sehr guten Freunden zähle. Hier möchte ich als erstes Herrn Dr. Maurice Scheer danken, mit dem die Zusammenarbeit beflügelte und das gemeinsame Projekt zu der interessantesten Zeit innerhalb der Doktorarbeit zu zählen ist. Auch geht mein Dank an meine beiden Bürokollegen Frau Dr. Anja Schulz und Herrn Dr. Constantin Bannert. Die Gespräche mit Euch waren immer toll und besaßen eine sehr kreative Note. Kann mir keine tolleren Bürogefährten wünschen! Ebenfalls ein besonderer Dank gilt Frau Maren Lang, die durch ihre sehr guten Kenntnisse in den Programmiersprachen sowie ihr umfangreiches biochemisches Wissen immer wieder mit Rat und Tat zur Seite stand.

Einen großen Dank geht auch an Herrn Dr. Karsten Hiller, durch den ich von der Biotechnologie zur Bioinformatik gekommen bin, an Frau Dr. Antje Chang, für ihre wertvollen Ratschläge und Unterstützung und an Herrn Dr. Michael Stelzer, der durch seine tolle Art eine wertvolle Bereicherung war und ist.

Insgesamt möchte ich allen Mitarbeitern der Abteilung für Bioinformatik und Biochemie der Technischen Universität Braunschweig für die wertvolle Unterstützung und das nette Arbeitsklima danken.

Kurzfassung

Die Analyse und Interpretation von metabolischen Netzwerken ermöglicht ein fundamentales Verständnis biologischer Systeme. Als interdisziplinäre Forschungsrichtung widmet sich die Systembiologie genau dieser Aufgabenstellung. Konzepte und Ansätze aus den „omik“ Teildisziplinen liefern ein integriertes Bild regulatorischer Prozesse. Dieses dient als Grundlage, um die komplexen Abläufe in Zellen von Organismen zu verstehen. So wachsen stetig das Detailwissen und die Datenmenge über Stoffwechselreaktionen, wobei eine Bearbeitung ohne bioinformatische Methodik nicht durchführbar wäre. Eine Abstraktion dieser Daten in Form eines Modells und die visuelle Darstellung ermöglichen oft einen zusätzlichen Erkenntnisgewinn.

Die BRaunschweig ENzyme DAtabase (BRENDA) stellt eine umfassende und weltweit wichtige Datenquelle für Informationen zu Enzymen und Stoffwechselwegen dar. So wurde auf dieser Datengrundlage eine generische Gesamtstoffwechselkarte entwickelt (Quester und Schomburg, 2011), welche umfangreiche Detailinformation zu Stoffwechselwegen bietet.

In der vorliegenden Dissertation wurde ein ganzheitliches System von Applikationen entwickelt, mit dem der Anwender metabolische Netzwerke visualisieren, editieren und analysieren kann.

Zu diesem Applikationssystem zählt zum einen der Pathway-Editor MapOmnia, der die Visualisierung und Bearbeitung von Netzwerken und im Speziellen die von metabolischen Netzwerken erlaubt. Um einen intuitiven Umgang zu gewähren, unterstützt die Applikation anwendungstypische Funktionen wie etwa „Drag and Drop“ (deutsch: Ziehen und Ablegen) sowie „Copy and Paste“ (deutsch: Kopieren und Einfügen), hat zahlreiche Ausrichtungs-Funktionen implementiert und unterstützt viele verschiedene Dateiformate für den Datenaustausch. Eine Schnittstelle zu etablierten Datenbanken erlaubt es, auf Informationen zu Enzymen und Stoffwechselwegen von BRENDA zuzugreifen. Es wurde somit ein grafisches Interface zu den Enzymen, Reaktionen, Metaboliten von BRENDA geschaffen. Die Darstellung ist auch bei großen Netzwerken performant möglich.

Die Dissertation stellt Konzepte und Methoden vor, die Anwender in die Lage versetzen, flexibel und intuitiv mit metabolischen Netzwerken zu arbeiten und diese zu analysieren.

Ein zentrales Ziel ist hierbei, Daten aus den Teildisziplinen Genomik, Transkriptomik, Proteomik und Metabolomik auf erstellte Interaktions-Netzwerke abzubilden. Die Implementierung eines Modellierungswerkzeuges erlaubt dem Anwender zudem, neue Netzwerke einfach aus etabliertem Wissen generieren zu können.

Anwender erlangen ein tiefergehendes Verständnis biochemischer Vorgänge durch implementierte Analysemethoden, wie etwa „Shortest Path“ (deutsch: kürzeste Wege) Berechnungen. Eine Skript-Schnittstelle ermöglicht darüber hinaus eine Analyse der Netzwerke im hohen Maße interaktiv zu erhalten, da Funktionalitäten und Methoden des Programms innerhalb von Skripten zur Laufzeit genutzt werden können, um beispielsweise Aufgaben zu automatisieren, Abfragen durchzuführen oder Netzwerke zu erstellen.

Durch Bereitstellung eines TCP/IP-Servers wurde eine Schnittstelle etabliert, welche auf die vielseitigen Anwendungsmöglichkeiten von MapOmnia zugreifen kann. So erlaubt dieses Konzept die Umsetzung der Webpräsenz BRIME (BRaunschweig Interactive Metabolism Explorer), die einen intuitiven Zugang zu metabolischen Netzwerken aus BRENDA bietet. Diese Webapplikation stellte die Suche von Substanzen, Enzymen sowie gesamter Reaktionen und Teilreaktionen bereit. Der Anwender kann metabolische oder andere „–ome“ Daten auf die generische Gesamtstoffwechselkarte abbilden, aber auch die netzwerkbasierte Visualisierung von Flussdaten, gewonnen aus der „Flux Balance Analysis“, wird unterstützt.

Abstract

The analysis and interpretation of metabolic networks provides a fundamental understanding of biological systems. As an interdisciplinary research direction, the Systems Biology is devoted precisely this task. Concepts and approaches from the "omics" disciplines provide an integrated picture of regulatory processes. This serves as a basis to understand the complex processes in cells of organisms.

Detailed knowledge and the amount of data about metabolic reactions are steadily growing. A processing without bioinformatic methods would not be feasible. An abstraction of the data in the form of a model and the visual representation often allows gaining an improvement of knowledge.

The BRaunschweig ENzyme DAtabase (BRENDA) is a comprehensive and important online source of information about enzymes and metabolic pathways. Based on these data, a generic metabolic map was developed (Quester and Schomburg, 2011), which provides extensive information about metabolic pathways.

In the present thesis, a holistic system of applications has been developed which allows the user to visualize, edit and analyze metabolic networks.

The pathway editor MapOmnia is one application of this system, which allows the visualization and manipulation of networks, and in particular of metabolic networks. To give an intuitive handling, the application supports application-typical functions such as "drag and drop" and "copy and paste". It provides numerous layout functions and supports many different file formats for data exchange. An implemented database-interface allows access to information on enzymes and metabolic pathways of BRENDA. Thus a graphical interface to the enzymes, reactions, metabolites of BRENDA was created. The presentation is fast, even for large networks.

This thesis presents concepts and methods in order to enable users to work flexibly and intuitively with metabolic networks and to analyze them. A key goal of this thesis is, to map data from the sub-disciplines of genomics, transcriptomics, proteomics and metabolomics onto generated interaction networks. The implementation of a modeling tool allows the user to easily generate new networks from established knowledge.

Users gain a deeper understanding of biochemical processes by using implemented analysis methods, such as "Shortest Path" calculations. A scripting interface also allows the interactive analysis of networks. Functions and methods of the program can be used within scripts at run time, for example, to automate tasks, to perform queries and to create networks.

An interface was established by providing a TCP/IP-server which can access the versatile functions of MapOmnia. This concept allows the implementation of BRIME (BRaunschweig Interactive Metabolism Explorer) website that offers an intuitive approach to metabolic networks from BRENDA. This web application provides an integrated search functionality to search substances, enzymes and total- and partial-reactions. The user can map metabolic or other "-ome" data to the generic overall map, but also the network-based visualization of flow data obtained from the "Flux Balance Analysis", is supported.

Abkürzungsverzeichnis

2D	2 Dimensional
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATP	Adenosintriphosphat
BioPAX	Biological Pathways Exchange
BMP	Bitmap
BMP	Bitmap
BRENDA	BRaunschweig ENzyme DAtabase
BRIME	BRaunschweig Interactive Metabolism Explorer
CPU	Central Processing Unit
CSV	Comma-Separated Values
DNA	Deoxyribonucleic acid
DTD	Document Type Definition
EC	Enzyme Commission
ECMA	European Computer Manufacturers Association
EPS	Encapsulated PostScript
ESI	Elektrospray-Ionisation
FBA	Flux Balance Analysis
FIFO	First–In–First–Out
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
GIF	Graphic Interchange Format (optional)
GML	Graph Modelling Language
GO	Gene Ontology
GPGL	6-Phosphogluconelactone
GPL	GNU General Public License
GraphML	Graph Markup Language
GSL	GNU Scientific Library
GUI	Graphical User Interface

GWT	Google Web Toolkit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
INI	Initialization
IP	Internet Protocol
IUBMB	International Union of Biochemistry and Molecular Biology
JPEG	Joint Photographic Experts Group
JPG	Joint Photographic Experts Group
KEGG	Kyoto Encyclopedia of Genes and Genomes
KGML	KEGG Markup Language
LaTeX	Lamport TeX
LGPL	GNU Lesser General Public License
LIFO	Last–In–First–Out
MALDI-TOF	Matrix-assisted laser desorption/ionization - Time of Flight
MDI	Multiple Document Interface
MDL	Molecular Design Limited Molfile
mRNA	messenger RNA
MVC	Model–View-Controller
NAD	Nicotinamid-Adenin-Dinukleotid
NMR	Nuclear magnetic resonance
PATRIC	PathoSystems Resource Integration Center
PBM	Portable Bitmap
PDF	Portable Document Format
PGM	Portable Graymap
PHP	PHP: Hypertext Preprocessor, früher: Personal Home Page Tools
PNG	Portable Network Graphics
PPM	Portable Pixmap
PS	PostScript
RNA	Ribonucleic acid
RTF	Rich Text Format
SAX	Simple API for XML

SBML	Systems Biology Markup Language
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol und Internet Protocol
TIFF	Tagged Image File Format
W3C	World Wide Web Consortium
WWW	World Wide Web
XBM	X11 Bitmap
XGMML	eXtensible Graph Markup and Modeling Language
XML	Extensible Markup Language
XPM	X11 Pixmap

Inhaltsverzeichnis

1 Einleitung

Die Visualisierung von Daten dient der Darstellung von Informationen und nicht zuletzt einer Informationsgewinnung durch das menschliche Wahrnehmungssystem und den Interpretationsmöglichkeiten (Murrmann, 2008).

So ermöglicht diese Visualisierung oft ein tieferes Verständnis komplexer Zusammenhänge. Visualisierungen werden beispielsweise in der Medizin eingesetzt, um medizinische Diagnosen zu stellen. In der Industrie werden Prozessmodelle verwendet und dargestellt, um Abläufe oder Organisationseinheiten zu modellieren und zu optimieren.

Die Erstellung von Modellen, als Abstraktionsbasis realer Zusammenhänge, ist hierbei ein zentraler Ansatzpunkt für das Verständnis und die Analyse komplexer Vorgänge.

So war zunächst die Zerlegung komplexer Vorgänge in einfache über Jahrzehnte ein erfolgreiches Modell (Merkl und Waack, 2009). Dieser „Top-Down“ (deutsch: von oben nach unten)-Ansatz kann in Analogie zu der in der Informatik üblichen algorithmischen Vorgehensweise des reduktionistischen Lösungsansatzes „Divide and Conquer“ (deutsch: Teile und Herrsche) gesehen werden. Jedoch hat sich gezeigt, dass es essenziell ist, die erhaltenden Lösungen der Problemstellungen niedrigerer Komplexität innerhalb eines „Bottom-Up“ (deutsch: von unten nach oben)-Ansatzes zu vereinheitlichen (Merkl und Waack, 2009). Durch diese Vorgehensweise kann das Gesamtproblem rekonstruiert und modelliert werden. Die Systembiologie widmet sich als interdisziplinäre Forschungsrichtung genau dieser Aufgabenstellung (Merkl und Waack, 2009; Eckstein, 2011). Die Konzepte und Ansätze aus den „omik“ Teildisziplinen sollen ein integriertes Bild regulatorischer Prozesse liefern. Sie dienen hierbei als Grundlage, um die komplexen Abläufe in Zellen von Organismen zu verstehen. Die gewonnenen Erkenntnisse durch die Aufklärung des Genoms, des Transkriptoms, des Proteoms und des Metaboloms ermöglichen es, kausale Zusammenhänge zu erkennen und zu definieren (Lin und Qian, 2007). Hier ist das Ziel der Forschung, das Zusammenwirken von Metaboliten und Enzymen innerhalb komplexer metabolischer Netzwerke zu verstehen und mathematisch modellieren zu können (Kremling, 2011). Da das Detailwissen über Stoffwechselreaktionen in solchen Netzwerken stetig wächst und die Datenmenge durch die Etablierung von Hochdurchsatzmethoden ohne bioinformatische Methoden schwierig

zu bearbeiten wäre, wird die Weiterentwicklung solcher Werkzeuge vorangetrieben (Kitano, 2002). Hierdurch entsteht ein rekursives Zusammenspiel zwischen Modellierung und Experiment, um den Erkenntnisstand über diese komplexen Vorgänge weiter zu erhöhen. Dieses gilt ebenfalls für die Darstellung metabolischer Netzwerke. So können beispielsweise experimentell gewonnene Daten auf etablierten Netzwerken abgebildet werden, um dieses visuell zu analysieren und zu validieren (Kremling, 2011).

1.1 Ziele und Motivation

Im Rahmen dieser Arbeit sollen Konzepte, Methoden und Anwendungen etabliert werden, die den Fokus auf die Visualisierung und Editierung metabolischer Netzwerke legen. Diese Konzepte und Methoden sollen den Anwender in die Lage versetzen flexibel und intuitiv mit metabolischen Netzwerken zu arbeiten und diese zu analysieren.
Die Ziele dieser Arbeit sind in drei Unterkapitel gegliedert. In Kapitel 1.1.1 werden die Ziele für die Etablierung eines Editors erörtert, mit dessen Hilfe Anwender metabolische Netzwerke erstellen und bearbeiten können. Im weiteren Verlauf dieser Arbeit wird dieser Editor auch als Pathway-Editor bezeichnet. Um erstellte Netzwerke im Web verfügbar zu machen, wurde ein Server etabliert. Die Konzeption und Umsetzung wird in Kapitel 1.1.2 beschrieben. Dieser Server dient als Interface für die Bedienung des Pathway-Editors. Kapitel 1.1.3 beschreibt die Anforderungen und Ziele, die für die Umsetzung der webbasierten Erkundung von metabolischen Netzwerken essenziell sind.

1.1.1 Pathway-Editor

Ziel dieses Teilprojektes dieser Arbeit ist die Implementierung eines Editors, mit dessen Hilfe metabolische Netzwerke nicht nur darstellbar, sondern ebenfalls erstell- und editierbar sind. Die Darstellung dieser Netzwerke muss hierbei performant erfolgen, um ein komfortables Arbeiten mit den Netzwerken zu ermöglichen. Editierfunktionen des Editors beziehen sich neben Operationen mit Graphen, wie etwa das Entfernen oder Hinzufügen von Netzwerkobjekten (Knoten und Kanten), ebenfalls auf die Veränderungsmöglichkeiten in der visuellen Darstellung der Netzwerke.
Der Editor soll ein breites Spektrum anwendungstypischer Funktionen unterstützen. Hierzu gehören beispielsweise „Drag and Drop“ (deutsch: Ziehen und Ablegen) sowie „Copy and Paste“ (deutsch: Kopieren und Einfügen) von einzelnen Knoten sowie gesamter selektierter

Unternetzwerke. Diese Funktionalitäten ermöglichen einen intuitiven und komfortablen Umgang mit dem Editierwerkzeug.

Im Weiteren sollen Layout-Funktionen implementiert werden, wobei zusätzlich Normalisierungsoptionen in definierbaren Teilbereichen, wie etwa das Angleichen von Abständen, das Verschieben auf Rasterpunkte oder die Skalierung und die Rotation dieser Teilselektion auswählbar sein sollen. Die Möglichkeit, benutzerfreundlich Gruppierungen von Knoten und Kanten vorzunehmen, um etwa Stoffwechselwege abzugrenzen sowie das Zusammenführen von Netzwerken ist geplant. Diese Konzepte erlauben effiziente Manipulation von Netzwerken sowie die individuelle Anpassung metabolischer Netze.

Die Notwendigkeit, die generierten Datenmengen aus den Teildisziplinen Genomik, Transkriptomik, Proteomik und Metabolomik auf erstellte Interaktions-Netzwerke abbilden zu können, ist ein zentrales Ziel der Softwareapplikation. Hierbei soll die Abbildung experimentellen Daten zu Veränderungen der Eigenschaften bei Kanten, Knoten und Gruppen führen.

Der Pathway-Editor soll zahlreiche Dateiformate für die Dateneingabe und -ausgabe unterstützen. Dies ermöglicht eine Integration bestehender Daten.

Die Applikation soll eine Schnittstelle zu etablierten Datenbanken besitzen, welche Informationen zu Enzymen und Stoffwechselwege anbieten, um bereits probate Stoffwechselkarten laden zu können. Weiterhin wird die Implementierung eines Modellierungswerkzeuges angestrebt, um neue Netzwerke einfach aus etabliertem Wissen generieren zu können.

Die Softwareapplikation soll Analysemethoden, wie etwa „Shortest Path“ (deutsch: kürzeste Wege) Berechnungen, beherbergen. Die Visualisierung kürzester Wege erlaubt es, tiefergehendes Verständnis biochemischer Vorgänge innerhalb des metabolischen Netzwerkes zu erlangen. Um die Analyse der Netzwerke im hohen Maße interaktiv zu erhalten, soll eine Skript-Funktionalität implementiert werden, durch die der Benutzer mit den erzeugten Netzwerken und darin befindlichen Netzwerkobjekten kommunizieren oder selber Netzwerke erstellen kann.

Ein hoher Fokus wird auf die Selektions-, Bearbeitungs-, Navigations- sowie Sortier- als auch Filtermöglichkeiten für Knoten, Kanten, Gruppen und Teilnetzwerken gelegt. Dies soll zu einer hohen Benutzerfreundlichkeit im Umgang mit erzeugten Netzwerken beitragen.

1.1.2 Pathway-Server

Die Erstellung eines Pathway-Editors, mit dessen Hilfe metabolische Netzwerke darstellbar und editierbar sind, bietet zahlreiche Konzepte und Lösungsansätze für Problemstellungen, welche sich aus der Visualisierung, der Analyse und Erstellung von metabolischen Netzwerken ergeben. Hierzu gehören neben der performanten Darstellung der Netzwerke auch die benutzerspezifischen Schnittstellen für Operationen mit Graphen, Import- und Export-Schnittstellen aber auch implementierte Analysemethoden für Graphen, wie etwa Graphsuchalgorithmik oder das Filtern auf Datenvorkommen innerhalb von Netzwerken.

Um diese implementierten Lösungsmöglichkeiten auch außerhalb des Editors zu nutzen, soll hierfür eine Kommunikationsschnittstelle in Form eines TCP/IP-Servers etabliert werden, die auf Ressourcen und Methoden des Pathway-Editors zugreifen und diese nutzen kann. Somit wird ein Server-Programm etabliert, welcher einen Zugang zu speziellen Diensten des Pathway-Editors offeriert und dessen Kommunikation nach dem Client-Server-Modell erfolgen kann. Dieses Projekt soll im Weiteren als Schnittstelle für den webbasierten Stoffwechselweg-Betrachter dienen.

1.1.3 Pathway-Webprojekt

In dem Institut Bioinformatik und Biochemie der Universität Braunschweig wurde eine generische Gesamtstoffwechselkarte entwickelt (Quester und Schomburg, 2011), welche umfangreiche Detailinformation zu Stoffwechselwegen bietet. Um einen webbasierten Zugang zu dieser Stoffwechselkarte zu ermöglichen, soll eine Webpräsenz erstellt werden, mit deren Hilfe die manuell erstellte Stoffwechselkarte, ähnlich der Google Maps-Technologie, dargestellt werden kann.

Es soll somit ein grafisches Interface zu den Enzymen, Reaktionen, Metaboliten von BRENDA (BRaunschweig ENzyme DAabase) geschaffen werden. BRENDA stellt eine umfassende und weltweit wichtige Datenquelle für Informationen zu Enzymen und Stoffwechselwegen dar (Scheer *et al.*, 2011). Neben der Gesamtstoffwechselkarte sollen zahlreiche Netzwerke von Organismen angeboten werden. Bei der Umsetzung soll eine interaktive und intuitive Explorations- und Navigationsmöglichkeit im Vordergrund stehen. Diese Umsetzung erlaubt dem Benutzer diese Webpräsenz intuitiven Umgang mit dem hohen Informationsgehalt der metabolischen Netzwerke.

Zusätzlich soll eine Benutzerschnittstelle für die Suche von Substanzen, Enzymen sowie gesamter Reaktionen und Teilreaktionen bereitgestellt werden. Diese Such- und Filterfunktion erlaubt dem Nutzer, individuelle Fragestellungen zu beantworten. Ein weiteres Ziel des Projekts ist die Möglichkeit, benutzerfreundlich metabolische oder andere „–ome“ Daten auf die generische Gesamtstoffwechselkarte abzubilden und diese als grafisches Merkmal der Netzwerkobjekte darzustellen. Hierbei soll die Abbildung der Daten wie im Pathway-Editor-Teilprojekt zu Veränderungen der Eigenschaften von Kanten und Knoten führen. Auch die netzwerkbasierte Visualisierung von Flussdaten gewonnen aus der „Flux Balance Analysis“ (deutsch: Fluss-Bilanz-Analyse; FBA), welche den Metabolitenfluss in metabolischen Netzwerken analysiert (Orth *et al.*, 2010), soll durch das Webprojekt unterstützt werden.

Die Erstellung der Webpräsenz ist ein Gemeinschaftsprojekt, wobei die Suchfunktionalität auf Datenbasis von BRENDA durch Melanie Busch realisiert, die Schnittstelle zu den Organsimen durch Maurice Scheer etabliert wurde. Die Erstellung der Webseite und die Kommunikation mit dem Pathway-Server ist Teil dieser Arbeit.

1.2 Struktur der Arbeit

Da es sich bei der Arbeit um drei Teilprojekte handelt, gibt dieses Kapitel eine kurze Übersicht über die Struktur der Arbeit. Dieses ermöglicht, den integralen Zusammenhang der Teilprojekte ganzheitlich widerzuspiegeln.

Die vorliegende Arbeit teilt sich in 5 Kapitel. In Kapitel 2 werden wichtige Grundlagen erörtert, die zum Verständnis der Arbeit notwendig sind. Kapitel 2.1 widmet sich den Grundlagen der Informatik, wobei Definitionen und Aspekte der Graphentheorie sowie die Visualisierung von Graphen und Daten erörtert werden. Im zweiten Teil der Grundlagen (Kapitel 2.2) wird eine Einführung in die Systembiologie gegeben. Hier werden die Funktionen von Enzymen (Kapitel 2.2.2) und Metaboliten (Kapitel 2.2.3) beschrieben sowie ihre Rolle innerhalb biochemischer Reaktionen (Kapitel 2.2.8) und den daraus resultierenden Stoffwechselwegen (Kapitel 2.2.9) und metabolischen Netzwerken (Kapitel 2.2.10) dargestellt. Die Kapitel 2.2.4 bis Kapitel 2.2.7 widmen sich den vier Vertretern der „omik“ Disziplinen Genomik, Transkriptomik, Proteomik und Metabolomik.

In Kapitel 3 werden das verwendete Material sowie die verwendeten Methoden aufgezeigt. Als Erstes wird die Entwicklungsumgebung (Kapitel 3.1) spezifiziert. Nach einer

Beschreibung der verwendeten Programmiersprachen (Kapitel 3.2) sowie Auszeichnungssprachen (Kapitel 3.3) werden die Grundlagen der Datenbanksysteme (Kapitel 3.4) erörtert. Im Anschluss wird das TCP/IP-Protokoll (Transmission Control Protocol und Internet Protocol), als Protokoll-Kombination aus dem TCP (Transmission Control Protocol) und dem IP (Internet Protocol), beschrieben, welches die Kommunikationsgrundlage des Pathway-Servers darstellt. Sämtliche Programme (Kapitel 3.6) sowie Bibliotheken und Frameworks (Kapitel 3.7), die zur Erstellung der Softwareapplikationen Verwendung fanden, werden anschließend beschrieben. Da die etablierten Applikationen unterschiedliche Dateiformate lesen und schreiben können, findet man in Kapitel 3.8 eine Übersicht über diese Formate. Abschließend wird in Kapitel 3.9 ein Überblick über die verwendeten Datenquellen gegeben. Diese waren eine notwendige Grundlage für die Etablierung der Stoffwechselkarte und finden ebenfalls Verwendung für die Abfragefunktionen metabolischer Daten in den einzelnen Programmen.

In Kapitel 4 werden die Ergebnisse der Projekte MapOmnia, MapNet-Server und BRIME vorgestellt, und ihr Leistungsumfang dokumentiert. Hierbei ist MapOmnia der Name des Pathway-Editors, MapNet der Name des Pathway-Servers und BRIME der Name des Webprojektes. BRIME steht für BRaunschweig Interactive Metabolite Explorer.

Zunächst werden in Kapitel 4.1 die Ergebnisse von MapOmnia präsentiert. Nach Aufzeigen der Programmmetrik (Kapitel 4.1.1) und der Dokumentation durch Doxygen (Kapitel 4.1.2) wird die Benutzeroberfläche des Pathway-Editors (Kapitel 4.1.3) präsentiert. Im Anschluss folgen Beschreibungen wichtiger Klassen von MapOmnia (Kapitel 4.1.4). In Kapitel 4.1.5 „Komponenten und Bedienung“ wird die Funktionalität des Pathway-Editors behandelt. Die angebotenen Layout-Funktionen werden in Kapitel 4.1.6 aufgeführt. Abschließend folgen Anwendungsbeispiele ausgesuchter Programmeigenschaften (Kapitel 4.1.7) und ein Vergleich mit anderen Pathway-Editoren wird aufgezeigt.

In Kapitel 4.2 wird beschrieben, wie der Pathway-Server realisiert und implementiert wurde, der als Bindeglied zwischen den beiden Hauptprojekten MapOmnia und BRIME angesehen werden kann. Nach kurzer Übersicht der Programmmetrik (Kapitel 4.2.1) und der Dokumentation durch Doxygen (Kapitel 4.2.2) werden auch hier wichtige Klassen des

Servers (Kapitel 4.2.3) präsentiert. Im Anschluss wird die Kommunikation des Servers (Kapitel 4.2.4) gezeigt.

In Kapitel 4.3 werden die Ergebnisse von BRIME präsentiert. Auch hier werden kurz statistische Parameter der Programmmetrik (Kapitel 4.3.1) von BRIME aufgeführt, die Dokumentation (Kapitel 4.3.2) beschrieben und einen Überblick über das Erscheinungsbild von BRIME (Kapitel 4.3.4) gegeben. Eine ausführliche Betrachtung von BRIME erfolgt in Kapitel 4.3.5. Kapitel 4.3.6 zeigt die grafischen Anpassungsmöglichkeiten der Netzwerkobjekte. Abschließend werden auch hier Anwendungsbeispiele von BRIME (Kapitel 4.3.7) gezeigt und ein Vergleich zu anderen metabolischen Web-Browser (Kapitel 4.3.8) durchgeführt.

Das letzte Kapitel 5 fasst die Ergebnisse der Arbeit des Pathway-Editors MapOmnia, der Webpräsens BRIME und des Pathway-Servers MapNet zusammen und gibt einen Ausblick auf sinnvolle und mögliche Erweiterungen.

2 Grundlagen

In diesem Kapitel werden die nötigen Grundlagen im Bereich der Informatik (Kapitel 2.1) und der Systembiologie (Kapitel 2.2) erörtert, welche essenziell für das Verständnis der vorliegenden Arbeit sind.

2.1 Informatik

2.1.1 Einführung in die Graphentheorie

In dieser kurzen Einführung in die Graphentheorie werden Definitionen (Kapitel 2.1.2), Operationen (Kapitel 2.1.3) sowie elementare Graphsuchalgorithmen (Kapitel 2.1.4) vorgestellt. Abschließend wird aufgezeigt, wie Daten auf Graphen abgebildet werden können (Kapitel 2.1.5). Es sei erwähnt, dass diese Einführung nicht eine vollständige Abhandlung der Definitionen und Grundlagen von Graphen beinhaltet. Für eine detaillierte Übersicht sei auf das Buch „Graphentheorie- Eine anwendungs-orientierte Einführung“ (Tittmann, 2003) verwiesen.

2.1.2 Definitionen

Diese Arbeit beschäftigt sich maßgeblich mit der Darstellung, Manipulation und Visualisierung metabolischer Netzwerke. Da ein metabolisches Netzwerk ebenso wie andere netzartige Strukturen wie soziale Netzwerke, Computernetze oder auch chemische Moleküle einem mathematischen Modell, dem Graphen, zugrunde liegt, werden im Folgenden grundlegende Definitionen vorgestellt (Tittmann, 2003).

Ein Graph ist ein Paar G= (V, E) disjunkter Mengen. Hierbei sind die Elemente von E eine zweielementige Teilmenge von V mit $E \subseteq [V]^2$. V nennt man Knoten, E Kanten des Graphen G (Diestel, 2006). Kanten verbinden zwei Knoten und werden in der Form e = {u, v} beschrieben. Fallen die Endknoten zusammen so wird die Kante als *Schlinge* bezeichnet. *Parallele Kanten* e = {u, v} und f = {u, v} haben dieselben Endknoten. Graphen ohne Schlingen und parallele Kanten werden als *schlichte Graphen* bezeichnet (Tittmann, 2003).

Graphen, bei denen die Menge der Knoten und Kanten endlich ist, heißen *endliche Graphen*. Ist dieses nicht der Fall, so heißen sie *unendlich* (Tittmann, 2003). Im Weiteren unterscheidet man zwischen gerichteten und ungerichteten Graphen.
In einem gerichteten Graphen besitzt jede Kante einen Anfangs- und Endknoten. Sie weisen somit einen Richtungssinn auf. In Analogie zu parallelen Kanten können Kanten in gerichteten Graphen auch antiparallel zueinander sein. Abbildung 2.1 zeigt exemplarisch einen gerichteten sowie ungerichteten Graphen.

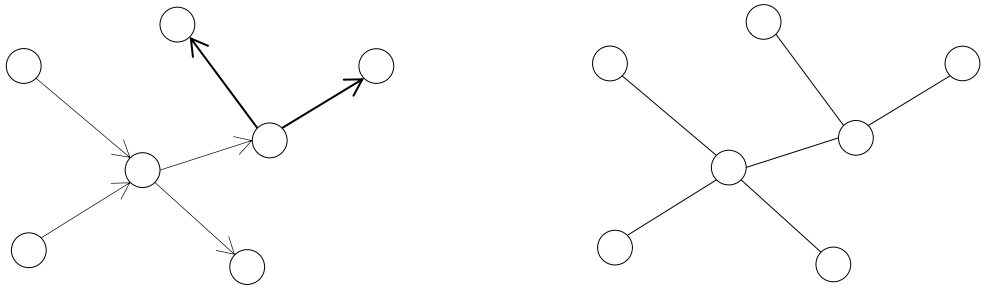

Abbildung 2.1: Der linke Graph zeigt exemplarisch einen gerichteten, der rechte einen ungerichteten Graphen.

Graphen können auf charakteristische Parameter untersucht werden. Hier sind beispielsweise die Knotenzahl, welches die Anzahl der Knoten in einem Graphen ist, sowie die Kantenzahl, welches dementsprechend die Anzahl der Kanten bezeichnet, zu betiteln. Ebenso besitzt ein Graph einen Minimalgrad sowie Maximalgrad (Kapitel 2.1.2.1), eine Taillenweite und Durchmesser (Kapitel 2.1.2.2) als auch eine Knotenzusammenhangszahl und eine Kantenzusammenhangszahl (Kapitel 2.1.2.3).

2.1.2.1 Knotengrade

Knoten, die durch eine Kante verbunden sind, heißen adjazent oder benachbart. Betrachtet man einen Knoten in einem Netzwerk, werden alle zu ihm adjazente Knoten als Nachbarschaft bezeichnet. Die Anzahl der Nachbarn gibt den Knotengrad an, wobei Schlingen doppelt gezählt werden. Besitzt ein Knoten keine Nachbarn, ist sein Knotengrad Null und er wird als isoliert bezeichnet.

Der Minimalgrad gibt hierbei den Knotengrad an, welcher von allen Knoten des Netzwerks in seinem Grad den niedrigsten Wert aufweist. Der Maximalgrad gibt entsprechend den mit dem höchsten Knotengrad an.

2.1.2.2 Wege und Kreise

Innerhalb eines Netzwerkes können Kantenfolgen definiert werden. Eine Kantenfolge beginnt bei einem Knoten (Startknoten). Von diesem wird über eine Kante zu einem Nachbarn gesprungen, wobei dieser Knoten als neuer Starknoten fungiert. Ausgehend von dem neuen Startknoten wird weiter die Nachbarschaft betrachtet und somit sukzessive ein Weg im Graphen etabliert. Jeder Knoten darf allerdings nur einmal im Weg vorkommen. Die Anzahl der Kanten gibt die Länge des Weges an. Ist der Startknoten der Kantenfolge identisch mit dem Endknoten, so bildet diese Folge einen Kreis im Graphen.
Der Durchmesser eines Graphen ist hierbei der größte Abstand zwischen zwei Knoten. Als Taillenweite eines Graphen bezeichnet man die Länge des kürzesten Kreises, sofern der Graph mindestens einen Kreis besitzt. Ist dieses nicht der Fall, so ist die Taillenweite unendlich.

2.1.2.3 Zusammenhang

Ein Graph G= (V, E) heißt zusammenhängend, sofern zwischen je zwei Knoten u und v aus der Menge V ein Weg existiert und somit diese Knoten durch eine Kantenfolge des Graphen verbunden werden können.
Die Knotenzusammenhangszahl eines Graphen G ist die minimale Anzahl von Knoten, durch deren Entfernung aus dem Graph dieser nicht mehr zusammenhängend ist. Analog gilt dieses für die Kantenzusammenhangszahl eines Graphen G.

2.1.2.4 Teilgraph und Untergraph

Bei einem Graph G= (V, E) können lokale Gebiete innerhalb des Graphen definiert werden. So unterscheidet man zwischen Teilgraph und Untergraph (Diestel, 2006).
Man spricht von einem Untergraphen, sofern in ihm Knoten aus G vorhanden sind und zusätzlich sämtliche Beziehungen (Kanten) zwischen diesen Knoten beibehalten werden, welche auch in G zwischen den ausgewählten Knoten vorlagen. Ist dies nicht der Fall, so handelt es sich um einen Teilgraphen. Somit ist jeder Untergraph ein Teilgraph, jeder

Teilgraph jedoch nicht zwangsläufig ein Untergraph. Abbildung 2.2 zeigt hierfür jeweils ein Beispiel.

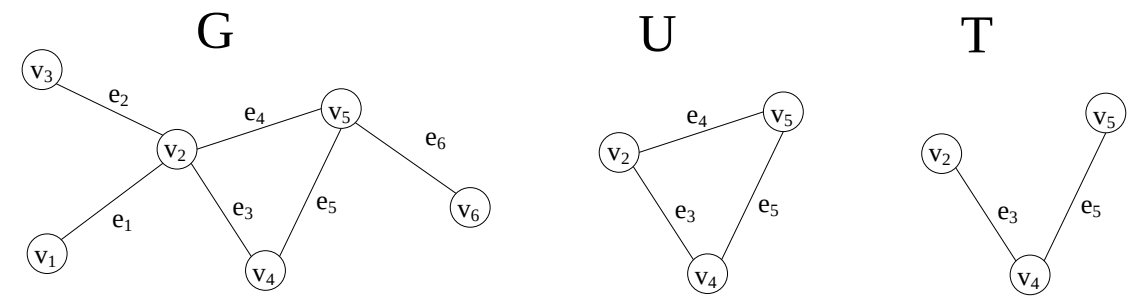

Abbildung 2.2: U ist ein Untergraph des Graphen G. T ist nur ein Teilgraph von G, da die Kante zwischen dem Knoten v_2 und dem Knoten v_5 fehlt.

2.1.2.5 Bipartiter Graph

Man spricht von einem bipartiter Graph Paar G= (V, E), sofern seine Knotenmenge V in zwei disjunkte Teilmengen zerfallen. Das heißt, dass nur zwei Gruppen von Knoten existieren und innerhalb dieser Teilgruppen keine Knoten durch Kanten miteinander verbunden sind.

2.1.3 Operationen mit Graphen

Operation mit Graphen beschreiben deren strukturelle Umformungen. Diese umfassen das Entfernen von Knoten und Kanten aus Graphen, die Fusion, Kontraktion sowie die Vereinigung von Graphen. Die Operationen werden in den folgenden Unterkapiteln näher betrachtet.

2.1.3.1 Entfernen von Knoten und Kanten

Bei der Entfernung eines Knotens v aus einem Graph G= (V, E) entsteht ein neuer Graph G_{neu}=G-v. Bei der Entfernung des Knotens müssen sämtliche Kanten, die zu diesem Knoten gehören, entfernt werden. Die Entfernung eines Knoten wird in Abbildung 2.3 (zweite Reihe links) exemplarisch dargestellt.

Wird aus einem Graphen Graph G= (V, E) eine Kante entfernt, so wird durch diese Operation ebenfalls ein neuer Graph G_{neu}=G-e erzeugt. In Abbildung 2.3 (zweite Reihe rechts) wird das Entfernen einer Kante verdeutlicht.

2.1.3.2 Fusion von Knoten

Bei der Fusion oder Verschmelzung zweier Knoten aus einem Graph G= (V, E) entsteht der Graph G_{uv}. Hierbei entsteht anstelle der beiden Knoten ein neuer Knoten. Dieser besitzt die Nachbarschaft der zu verschmelzenden Knoten. In Abbildung 2.3 (dritte Reihe links) ist die Fusion zweier Knoten illustriert.

2.1.3.3 Kontraktion von Kanten

Die Kontraktion einer Kante e={u, v} aus einem Graph G= (V, E) lässt den neuen Graphen G/e entstehen, bei dem die Kante e entfernt wurde. Außerdem werden die Knoten u und v fusioniert. Diese Kontraktion einer Kante wird in Abbildung 2.3 (dritte Reihe rechts) dargestellt.

2.1.3.4 Vereinigung und Schnitt von Graphen

Bei der Vereinigung zweier Graphen, Graph G_1= (V_1, E_1) und Graph G_2= (V_2, E_2), entsteht ein Graph, der die Knotenmenge V_1 u V_2 sowie die Kantenmenge E_1 u E_2 besitzt. In Abbildung 2.3 (unterste Reihe links) wird die Vereinigung zweier Graphen exemplarisch dargestellt.

Bei dem Schnitt zweier Graphen, Graph G1= (V_1, E_1) und Graph G2= (V_2, E_2), entsteht ein geschnittener Graph, welcher die Knotenmenge $V_1 \cup V_2$ sowie die Kantenmenge $E_1 \cup E_2$ besitzt. Dieses wird ebenfalls in Abbildung 2.3 (unterste Reihe rechts) präsentiert.

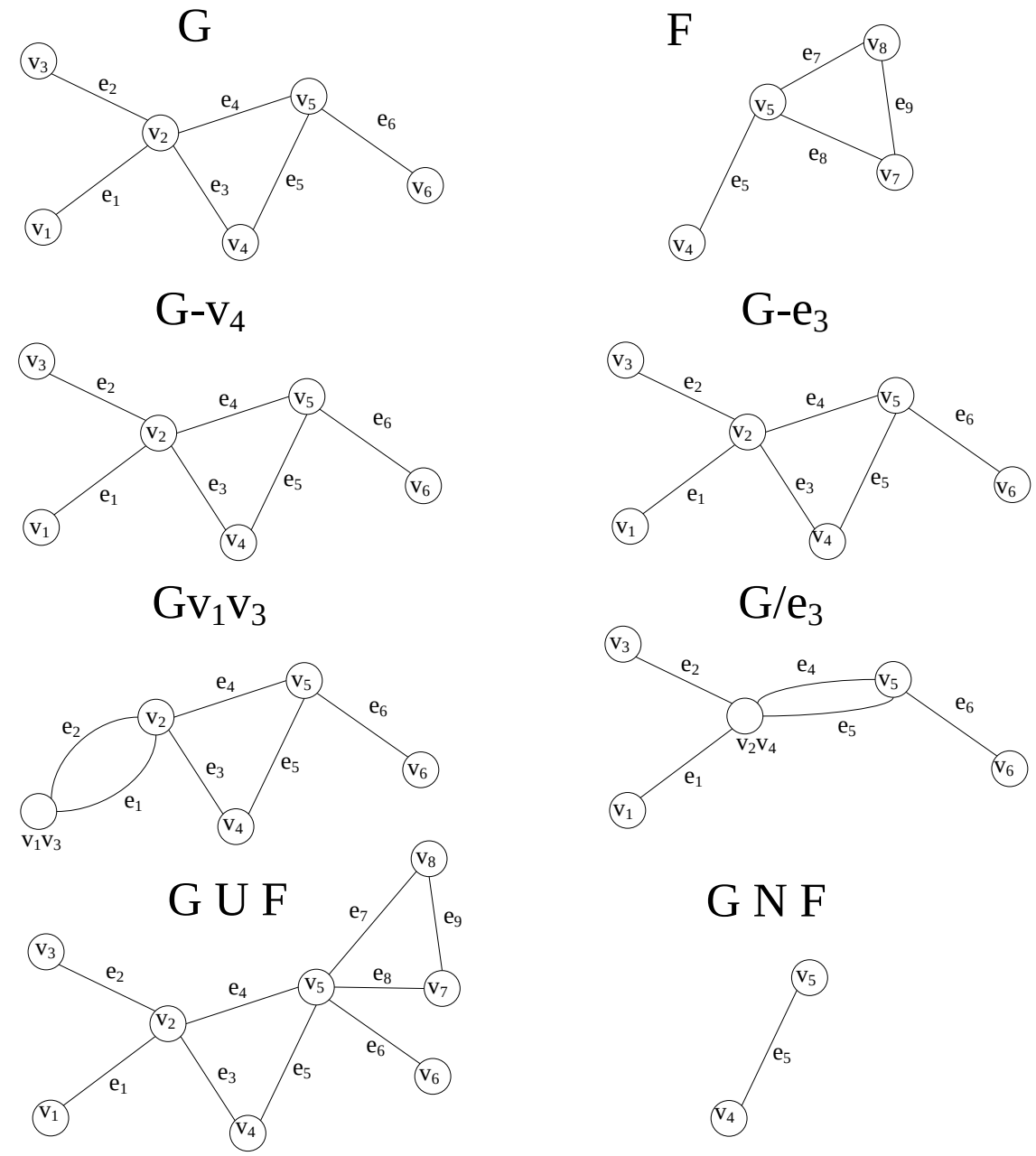

Abbildung 2.3: Operationen mit Graphen: In der ersten Reihe sind die Graphen G und F dargestellt. In der zweiten Reihe wurde auf der linken Seite der Knoten v_4 aus G entfernt und auf der Rechten die Kante e_3. Die Fusion der Knoten v_1 und v_3 von G lässt den Graphen in der dritten Reihe links entstehen. Die Kontraktion der Kante e_3 liefert den Graphen aus der dritten Reihe rechts. Bei der Vereinigung der Graphen G und F entsteht der Graph aus der untersten Reihe links und bei dem Schnitt dieser Graphen der Graph rechts.

2.1.4 Graphsuchalgorithmen

In den folgenden Unterkapiteln werden zunächst 2 Graphsuchalgorithmen, die Breitensuche (Kapitel 2.1.4.1) und die Tiefensuche (Kapitel 2.1.4.2), vorgestellt. Im Anschluss wird in Kapitel 2.1.4.3 der Dijkstra-Algorithmus (Dijkstra, 1959) beschrieben, welcher für das Auffinden kürzester Wege in der vorliegenden Arbeit Verwendung fand.

2.1.4.1 Breitensuche

Die Breitensuche ist ein Algorithmus für das Durchsuchen eines Graphen und stellt die gedankliche Grundlage vieler Graphalgorithmen, wie z. B. die Bestimmung des minimalen Spannbaums von Prim oder der Dijkstra-Algorithmus für das Finden kürzester Wege (Leiserson *et al.*, 2007).

Bei der Breitensuche werden ausgehend von einem Startknoten s über die Kanten seine Nachbarn erkundet. Die Knoten werden hierbei in drei Farbklassen eingeteilt. Weiße Knoten wurden noch nicht besucht. Dieses ist der Ausgangszustand aller Knoten zu Beginn der Suche. Graue Knoten wurden besucht, es muss allerdings überprüft werden, ob noch nicht erkundete Knoten (weiß) oder schwarze Knoten, welche komplett überprüft wurden, vorhanden sind. Der Startknoten s wird hierbei als besucht markiert. Für die Bearbeitungsreihenfolge der grauen Knoten wird eine Warteschlange eingesetzt. Diese hält die Objekte in linearer Ordnung und arbeitet nach dem FIFO-Prinzip („First–In–First–Out“). Somit wird der Startknoten in die Warteschlange platziert (Leiserson *et al.*, 2007).

Nach der Initialisierung wird solange aus dem Kopf der Warteschlange ein Knoten entnommen, solange sie nicht leer ist. Für den entnommenen grauen Knoten wird nun für alle Nachbarn überprüft, ob es sich dabei um Zielknoten handelt. Ist dieses der Fall, ist die Suche beendet. Andernfalls wird überprüft, ob der Knoten als nicht besucht (weiß) markiert ist. Ist dieses der Fall, wird dieser als besucht markiert und ebenfalls in die Warteschlange platziert (Leiserson *et al.*, 2007).

2.1.4.2 Tiefensuche

Analog zu der Breitensuche handelt es sich bei der Tiefensuche ebenfalls um einen Graphsuchalgorithmus. Es wird allerdings im Gegensatz zu der Breitensuche hierbei einem möglichen Pfad entlang, von dem Ausgangsknoten betrachtet, in die Tiefe gegangen. Wie

bei der Breitensuche gibt es drei Farbklassen: weiß, grau und schwarz mit derselben Bedeutung wie in Kapitel 2.1.4.1 (Leiserson *et al.*, 2007).
Für die Bearbeitungsreihenfolge der grauen Knoten wird ein Stapel eingesetzt, welcher die Objekte in linearer Ordnung hält und nach dem LIFO-Prinzip („Last–In–First–Out“ deutsch: zuletzt herein – zuerst hinaus) arbeitet (Leiserson *et al.*, 2007).
Zu Beginn sind alle Knoten weiß. Ausgehend von einem Startknoten s wird für alle seine Nachbarn überprüft, ob sie noch nicht besucht wurden. Ist dieses der Fall, so wird rekursiv für diesen Knoten die Tiefensuche aufgerufen, welches eine eigene Prozedur darstellt. In dieser wird zunächst überprüft, ob es sich um Zielknoten handelt. Trifft dieses zu, wäre die Suche beendet. Ansonsten wird der Stapel um die zu untersuchenden Knoten erweitert. Nun wird solange aus dem Stapel ein Knoten entnommen, solange er nicht leer ist und rekursiv die Tiefensuche aufgerufen (Leiserson *et al.*, 2007).

2.1.4.3 Dijkstra-Algorithmus

Innerhalb eines Graphen einen kürzesten Weg zwischen zwei Knoten des Graphen zu finden ist ein typisches Problem der Graphentheorie. Für die Lösung wird im Folgenden der Dijkstra-Algorithmus vorgestellt.
Ausgehend von einem Graph G= (V, E) mit gewichteten Kanten ist der kürzeste Weg zwischen einem Startknoten s und einem Endknoten e gesucht. Die Kantengewichte dürfen hierbei nicht negativ sein. Der Algorithmus verwaltet eine Knotenmenge S, für die eine endgültige Gewichtung der kürzesten Pfade vom Startknoten s aus schon bestimmt wurde. In jedem Schritt wählt der Algorithmus denjenigen Knoten u aus der Menge V-S mit der kleinsten Schätzung des kürzesten Pfades aus. Nun wird u zu S hinzugefügt und im Anschluss werden sämtliche Kanten von u ausgehend relaxiert, wobei jeder Nachbar v von u betrachtet wird, welcher in V-S vorhanden ist. Innerhalb dieser Relaxion wird überprüft, ob man dessen Nachbarn v von s aus über u besser (d. h. auf kürzerem Weg als bisher bekannt) erreichen kann (Leiserson *et al.*, 2007). Zu beachten ist, dass die Breitensuche ein Spezialfall eines Kürzesten-Wege-Problems darstellt, wobei hierbei alle Kanten ein Gewicht von 1 besitzen (Leiserson *et al.*, 2007).

2.1.5 Visualisierung von Daten

Graphen, welche die visualisiert dargestellt werden, eignen sich dazu Eigenschaften der Knoten und Kanten zu manipulieren, um Daten abzubilden. Hierbei bieten diese Netzwerkobjekte zahlreiche visuelle Attribute. Knoten haben eine räumliche Ausdehnung, und sie besitzen eine Größe, Kanten werden durch ihre Länge und Strichstärke repräsentiert. Man kann zusätzlich den Netzwerkobjekten Farben zuordnen. Im Allgemeinen werden genau diese Attribute (Größe und Farbe) genutzt, um Daten im Graphen zu visualisieren. In den folgenden Kapiteln werden kurz die unterschiedlichen Skalenniveaus, die Transformation der Daten und abschließend die Farb- sowie Größencodierung erörtert.

2.1.5.1 Skalenniveaus

Werden Daten von Merkmalen erhoben, so können auf dieses statistische Verfahren angewendet werden. Hierbei ist die Einteilung der Daten in verschiedene Skalenniveaus zu unterscheiden. Bei der Nominalskala werden Ausprägungen nur klassifiziert, es kann keine Reihenfolge gebildet werden. Bei der Ordinalskala kann zwar eine Reihenfolge bestimmt werden, jedoch ist der Abstand zwischen den Werten nicht quantifizierbar. Innerhalb einer Kardinalskala kann eine Rangfolge bestimmt werden, und die Abstände lassen sich durch ein Abstandsmaß erfassen. Hierbei sind die Skalenwerte reelle Zahlen (Internetdokument Skalenniveaus, 2005).

2.1.5.2 Transformation

In den folgenden zwei Kapiteln 2.1.5.3 und 2.1.5.4 werden zwei typische Visualisierungsmethoden, die Farbcodierung und die Größencodierung, vorgestellt, um Daten innerhalb eines Graphen darzustellen. Hierfür ist grundsätzlich eine Transformation der Daten nötig. Bei einer Transformation wird jeder Ausprägung des Merkmals ein neuer Wert zugeordnet. Das bedeutet, dass eine Überführung eines Wertes x in einen neuen Wert y nötig ist. Dieses wird durch eine Transformationsfunktion mit $y = f(x)$ ermöglicht.

2.1.5.3 Farbcodierung

Um Daten innerhalb eines Graphen darzustellen, kann die Farbe der Netzwerkobjekte durch eine Transformationsfunktion festgelegt werden. Hierdurch erfolgt eine Farbcodierung der abhängigen Werte, wobei unterschiedlicher Skalenniveaus verwendet werden können. Eine häufig verwendete Variante der Farbcodierung ist die Visualisierung der Werte durch eine Heatmap (Friendly, 1995). Der Betrachter ordnet den aus der Heatmap verwendeten Farben automatisch Temperaturen zu, wodurch eine intuitive Einteilung ermöglicht wird. Ebenfalls üblich sind Schwarz-Weiß- oder Grauwerte für die Durchführung einer Farbcodierung.

2.1.5.4 Größencodierung

Analog zu der Farbcodierung können Daten innerhalb eines Graphen durch die Veränderung der Größe der Netzwerkobjekte visualisiert werden. Hierbei sind für die Knoten die Höhe und Breite zugänglich. Für Kanten bietet sich ihre Strichstärke an, welche beispielsweise Verwendung findet, um metabolische Flüsse in einem Netzwerk darzustellen.

2.2 Systembiologie

In dieser Einführung in die Systembiologie wird das Konzept dieses noch recht jungen Zweigs der Biowissenschaften erörtert. Es wird die Funktionsweise von Enzymen (Kapitel 2.2.2) und die Bedeutung von Metaboliten (Kapitel 2.2.3) beschrieben. Anschließend wird ein Überblick über die 4 wichtigsten „omik“ Disziplinen, Genomik (Kapitel 2.2.4), Transkriptomik (Kapitel 2.2.5), Proteomik (Kapitel 2.2.6) und Metabolomik (Kapitel 2.2.7), gegeben. Die Bedeutung biochemischer Reaktionen (Kapitel 2.2.8) und der Aufbau von Stoffwechselwegen (Kapitel 2.2.9) sowie der Zusammenhang zu metabolischen Netzwerken (Kapitel 2.2.10) wird abschließend aufgezeigt.

2.2.1 Einführung in die Systembiologie

Biologische Systeme zeichnen sich durch eine Vielzahl von Komponenten aus, die miteinander verknüpft sind (Kremling, 2011). Somit unterliegen biologische Prozesse einer komplexen dynamischen Struktur. Um diese Systeme als Ganzes zu verstehen (Merkl und Waack, 2009), hat sich der Forschungsansatz in den Biowissenschaften geändert. So

widmet sich die Systembiologie genau dieser Erforschung biologischer Prozesse auf der Systemebene, wobei sie interdisziplinär Wissen und Technologien aus Mathematik, Informatik, Physik und Ingenieurwissenschaften verknüpft (Eckstein, 2011).

Moderne Analyseverfahren und Hochdurchsatztechnologien im Bereich der biochemischen Spezialdisziplinen generieren umfangreiche Datenmengen (Lin und Qian, 2007; Merkl und Waack, 2009). Hierbei wäre eine strukturierte Weiterverarbeitung dieser Daten ohne computerbasierte Verfahren nicht denkbar (Kitano, 2002). Zu den wichtigsten biochemischen Disziplinen, den „omik“ Disziplinen, gehören Genomik, Transkriptomik, Proteomik sowie Metabolomik (Lin und Qian, 2007).

Die Systembiologie stellt hier die mathematische Modellierung ins Zentrum der Aktivitäten (Kremling, 2011). So entstehen im gegenseitigen Austausch zwischen Modell und Experiment realitätsnahe Vorhersagen. Genau dieser iterative Zyklus aus Experiment und Modellierung begründet die besondere Leistungsfähigkeit der Systembiologie. Abbildung 2.4 zeigt diesen Zyklus.

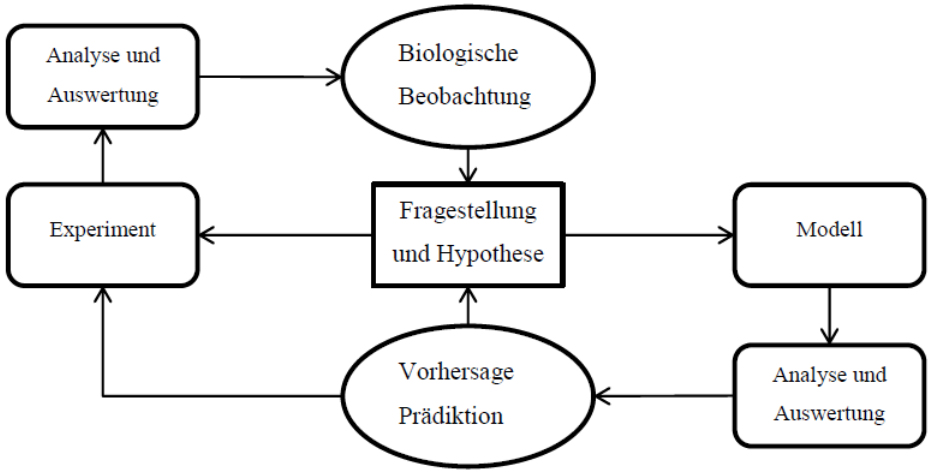

Abbildung 2.4: Iterativer Zyklus aus Fragestellung, biologischem Experiment und Modellierungsvorgang (Kremling, 2011).

Für die Analyse und Auswertung bedient man sich bioinformatischer Methodik, unter Verwendung etablierter Computerapplikationen. Für die Darstellung und Analyse metabolischer Netzwerke werden Pathway-Editoren verwendet. Dieses stellt das Betätigungsumfeld und die zentralen Aspekte der Applikationen, welche innerhalb dieser

Arbeit entwickelt wurden, dar. Im Folgenden werden die hierfür nötigen Grundlagen erörtert.

2.2.2 Enzym

Enzyme fungieren als Katalysatoren innerhalb biochemischer Reaktionen. Hierbei wird die Aktivierungsenergie herabgesetzt und die Reaktionsgeschwindigkeit dieser Prozesse deutlich erhöht. Enzyme ändern jedoch nicht die Gleichgewichtskonstante oder die freie Enthalpie der Reaktion. Enzyme arbeiten hochspezifisch und haben eine unterschiedliche Substratspezifität. Sie werden von Zellen und Mikroorganismen gebildet, wobei sie intra- und extrazellulär wirken können.

Neben der Substratspezifität zeichnen sich Enzyme auch durch eine Wirkungsspezifität aus. Diese beschreibt die Eigenschaft, dass jedes Enzym in der Regel nur eine definierte Veränderung des Substrates bewirkt.

Enzyme werden nach katalysierter Reaktion klassifiziert. Eine Übersicht bietet Tabelle 2.1. Enzyme werden nach einer sogenannten Enzym-Kommissions-Nummer (EC; Webb, 1992) der IUBMB systematisch benannt. Hierdurch ist eine eindeutige Zuordnung möglich. Diese Enzym-Kommissions-Nummer besteht aus 4 durch Punkte getrennten Zahlen. Die ersten drei Zahlen geben Klasse, Unter-Klasse und Unter-Unter-Klasse an, die vierte Zahl ist eine laufende Nummer.

Tabelle 2.1: Übersicht der sechs Enzymklassen, nach der IUBMB-Klassifikation

Klasse	Name	Beschreibung
1	Oxidoreduktasen	Elektronentransfer
2	Transferasen	Transfer chemischer Gruppen
3	Hydrolasen	Hydrolyse
4	Lyasen	Spaltung mit Abgang von chemischen Stoffen (nicht-hydrolytisch)
5	Isomerasen	Transfer von Gruppen innerhalb von Molekülen
6	Ligasen	Verbindung zweier Moleküle auf Kosten einer energiereichen Phosphatbindung (ATP Spaltung)

2.2.3 Metabolit

Bei Metaboliten handelt es sich um Substanzen, welche als Zwischenstufen bzw. als Abbauprodukte bei Stoffwechselvorgängen vorkommen. Je nach Betrachtungsweise lassen sich Metabolite in Substrate und Produkte bei biochemischen Reaktionen (vergleiche Kapitel 2.2.8) sowie in Haupt- und Nebenmetaboliten bei dem Metabolismus einteilen. Die Einteilung von Substrat und Produkt beruht hierbei auf der Reaktionsrichtung.
Nebenmetabolite kommen sehr häufig im Stoffwechsel vor (z. B. H2O, CO2, NADH, ATP). Für Hauptmetabolite lässt sich ein Stoffwechselweg definieren (vergleiche Kapitel 2.2.9). Sie werden von einer biochemischen Reaktion zu der nächsten weitergegeben, wobei sie durch Gruppenübertragungen, Abspaltungen oder Isomerisierungen modifiziert werden.

2.2.4 Genomik

Die wissenschaftliche Teildisziplin, die sich speziell mit dem Genom beschäftigt, heißt Genomik. Das Genom entspricht hierbei der Gesamtheit aller Gene eines Organismus. Innerhalb der Genomik wird die Bedeutung von Genen untersucht. Hierzu gehört z. B. der Einfluss auf das Wachstum und die Steuerung biologischer Systeme. So wird durch die Durchführung von Genomprojekten, wie etwa dem Humangenomprojekt, die Gesamtsequenz der DNA identifiziert und annotiert (Merkl und Waack, 2009).

2.2.5 Transkriptomik

Die Transkriptomik untersucht das transkriptionelle Profil einer Zelle zu einem definierten zeitlichen Zustand. Das Transkriptom wird durch die Menge an vorhandenen mRNA-Molekülen abgeleitet. Da Genprodukte dann transkribiert werden, wenn sie aufgrund einer spezifischen Situation benötigt werden, können durch Analyse der mRNA-Konzentration diejenigen Gene identifiziert werden, die zu diesem Zustand aktiviert wurden (Merkl und Waack, 2009).
Um das Transkriptom zu bestimmen, bedient man sich der Gen-Chip-Technologie, wobei die Gene auf einem Mikroarray identifiziert und deren Aktivität gemessen werden. Die Kombination von erstellten Modellen aus der transkriptionalen Regulation und metabolischen Netzwerken kann hierbei genutzt werden, um Vorhersagen über Phenotypisierung und Gen-Expression für ein Experiment zu tätigen. Auch können diese

Modelle benutzt werden, um Lücken sowie unbekannte Komponenten zu identifizieren und Interaktionen innerhalb des Netzwerkes aufzudecken (Covert *et al.*, 2004).

2.2.6 Proteomik

Im Bereich der Proteomik werden aktive Genfunktionen anhand von Proteinkonzentrationen abgeleitet (Merkl und Waack, 2009). Auch hier handelt es sich um eine Momentaufnahme des Organismus zu einem bestimmten Zeitpunkt. Da sich das Proteom aufgrund veränderter Bedingungen und äußerlichen Einflüssen, wie etwa der Temperatur, ständig verändert, ist es hochdynamisch in seiner Zusammensetzung, wobei Proteine synthetisiert und degradiert werden. Den Rückschluss auf eine aktive Genfunktion ist als sehr komplex zu betrachten, da Proteine zusätzlich posttranslational modifiziert werden und somit diese resultierende Vielfalt an Proteinen nur schwer mit biochemischen Methoden differenziert werden kann (Merkl und Waack, 2009).
Für die Bestimmung des Proteoms sind heute zwei Techniken üblich. Zum einen die 2 dimensionale Gel Elektrophorese (2D-PAGE) und zum anderen die Massenspektrometrie (MALDI-TOF, ESI-MS/MS) (O'Farrell, 1975; Wendisch *et al.*, 2006).

2.2.7 Metabolomik

Innerhalb der Metabolomik werden Stoffwechselprodukte in einer Zelle betrachtet (Fiehn, 2002; Oliver *et al.*, 1998). Hierbei gibt die Analyse des Metaboloms ein metabolisches Profil der Zelle unter definierten Bedingungen zurück. Mittels bioinformatischer Auswertung können Veränderungen im metabolischen Profil genutzt werden, um Rückschlüsse auf Reaktionen des Organismus auf genetische Veränderungen sowie umweltbedingte Einflüsse zu ziehen.
Um ein metabolisches Profil für Zellen aufzunehmen, stehen molekularbiologische Methoden wie die Kernspinresonanzspektroskopie (NMR), Infrarotspektroskopie (IR), Massenspektrometrie (MS) und weitere Chromatografieverfahren zu Verfügung.

2.2.8 Biochemische Reaktion

Innerhalb biochemischer Reaktionen spielen Enzyme die zentrale Rolle. Wie schon erwähnt, katalysieren sie eine biochemische Reaktion. Man kann eine Reaktion als Reaktionsgleichung darstellen, wobei nach Konvention auf der linken Seite Edukte und auf

der rechten Seite Produkte stehen. Der Reaktionspfeil gibt hierbei die Reaktionsrichtung vor. Bei Reaktionen, welche ebenfalls in die entgegengesetzte Richtung ablaufen können, werden zwei Pfeile verwendet. Bei der Rückreaktion wird das Produkt zum Edukt der Reaktion. Folgende Gleichung zeigt hierbei exemplarisch die Darstellung einer solchen Reaktionsgleichung:

Reaktionsgleichung 1: $Enzym\ 1\colon A + B \rightarrow C + D + E$

Für diese Reaktion kann auch eine grafische Repräsentation erfolgen, welche in der Abbildung 2.5 dargestellt ist.

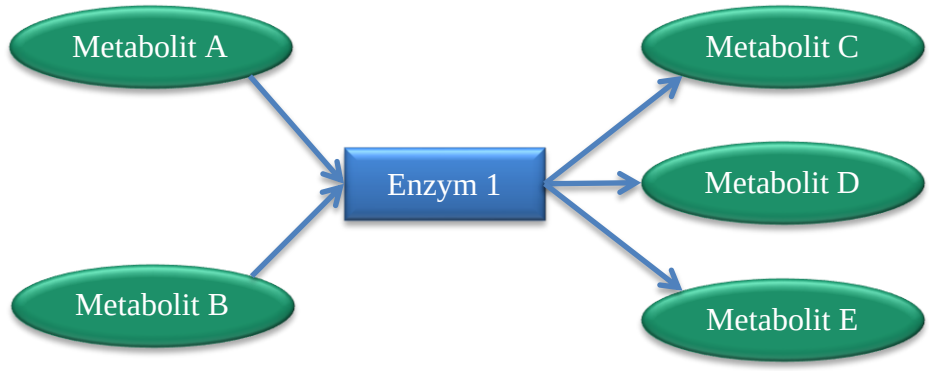

Abbildung 2.5: Eine Reaktion dargestellt als Graph: Metabolit A und B dienen als Edukte der Reaktion, die durch das Enzym 1 katalysiert wird. Die Produkte der Reaktion sind die Metabolite C, D und E.

2.2.9 Stoffwechselweg

Betrachtet man nicht nur eine biochemische Reaktion, sondern eine Abfolge solcher Auf-, Ab- und Umbauprozesse in einer Zelle, so entspricht dieses einem biologischen Prozess. Allerdings sind Stoffwechselwege nicht autark und sind untereinander stark vernetzt. Die folgende Reaktionsgleichung 2 erweitert die Reaktionsgleichung 1 aus Kapitel 2.2.8:

Reaktionsgleichung 1: $Enzym\ 1\colon A + B \rightarrow C + D + E$

Reaktionsgleichung 2: $Enzym\ 2\colon E + F \rightarrow G$

Diese Abfolge biochemischer Reaktionen lässt sich als grafisches Netzwerk visualisieren, welches in Abbildung 2.6 dargestellt ist.

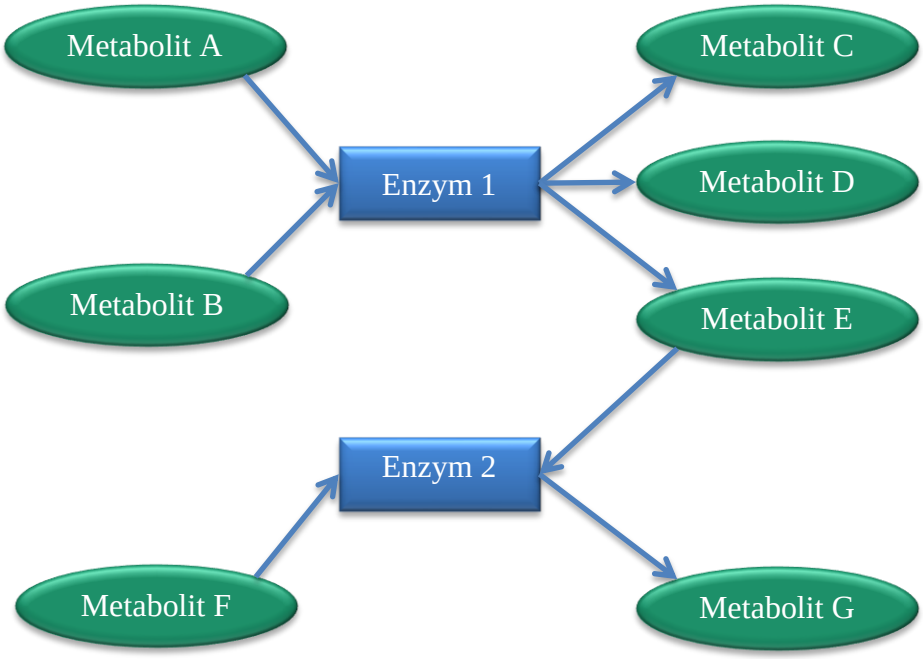

Abbildung 2.6: Schematischer Stoffwechselweg: Der Metabolit E, welcher aus der katalysierten Reaktion von Enzym 1 entsteht, wird direkt in einer weiteren Reaktion, katalysiert durch Enzym 2, zu den Metaboliten F und G verstoffwechselt.

2.2.10 Metabolisches Netzwerk

Durch die in den beschriebenen Einzeldisziplinen Genomik (Kapitel 2.2.4), Transkriptomik (Kapitel 2.2.5), Proteomik (Kapitel 2.2.6) und Metabolomik (Kapitel 2.2.7) gewonnenen Daten, besteht die Möglichkeit, ein biochemisches und molekularbiologisches Abbild der Zelle zu modellieren. Ein erklärtes Ziel der Kombination der Daten ist die Entwicklung der *in silico* Zelle, welcher ein mathematisches Modell zugrunde liegt. Wichtige Voraussetzung ist neben etablierten Hochdurchsatzverfahren auch die strukturierte Weiterverarbeitung der Daten mittels bioinformatischer Methoden, um die enorme Informationsflut über unterschiedliche Zustände der Zelle verarbeiten zu können.

Für die Grundlagenforschung der Beziehungen und Wechselwirkungen zwischen Metaboliten eignen sich Metabolomanalysen an Modellorganismen, wie z. B. *Escherichia coli* (Börner *et al.*, 2007) oder *Corynebacterium glutamicum* (Krömer *et al.*, 2005). Das molekulare und biochemische Netzwerk kann grafisch dargestellt werden und gibt eine detaillierte Übersicht der metabolischen Pfade innerhalb der Zelle. So können Aufbau-, Abbau- und Umwandlungsreaktionen nachvollzogen werden. Hierbei werden die biochemischen Reaktionen (Kapitel 2.2.8) in Stoffwechselwegen (Kapitel 2.2.9) gruppiert, um eine Wiedererkennung und biochemische Interpretationsmöglichkeit zu gewährleisten.

3 Material und Methoden

In diesem Kapitel wird ein umfassender Überblick über das verwendete Material sowie die verwendeten Methoden gegeben. Im folgenden Kapitel (Kapitel 3.1) wird zunächst die Entwicklungsumgebung spezifiziert. Anschließend wird eine kurze Einführung in die verwendeten Programmiersprachen (Kapitel 3.2) sowie Auszeichnungssprachen (Kapitel 3.3) gegeben. Der entwickelte Pathway-Editor MapOmnia sowie die Webapplikation BRIME verfügen über Datenbankschnittstellen für die Kommunikation zu relationalen Datenbanken. Die notwendigen Grundlagen werden in Kapitel 3.4 erörtert. In Kapitel 3.5 wird das TCP/IP-Protokoll beschrieben, welches die Kommunikationsgrundlage des TCP/IP Servers darstellt.
Die für die Realisierung notwendigen Programme (Kapitel 3.6) sowie Bibliotheken und Frameworks (Kapitel 3.7) werden aufgezeigt und erläutert. Die innerhalb der Arbeit benutzten Dateiformate werden in Kapitel 3.8 kurz beleuchtet und in Kapitel 3.9 wird ein Überblick über die verwendeten Datenquellen gegeben.

3.1 Entwicklungsumgebung

Die Entwicklung von MapOmnia sowie von BRIME fand ausschließlich in einer Linuxumgebung statt. Sämtliche benutzten Bibliotheken und Frameworks wurden allerdings so gewählt, dass eine Kompatibilität zu Windows vorhanden ist. Als Betriebssystem kam die 64 Bit Variante von Ubuntu zum Einsatz. Die zuletzt verwendete Version von Ubuntu war hierbei die Ubuntu Desktop 12.04 LTS Version.

3.1.1 Hardware

Die Entwicklung von MapOmnia und BRIME wurde auf zwei verschiedenen Rechnersystemen durchgeführt. Zum einen wurde ein Desktop PC mit Intel(R)- Corei7-920 Quad-Core Prozessor getaktet mit 3,20 GHz verwendet. Der Arbeitsspeicher umfasste hierbei 4 GB. Zum anderen wurde ein Notebook verwendet, welches mit einem Intel(R)-Core(TM)2 Duo T6500 Prozessor getaktet mit 2,10 GHz und 4 GB Arbeitsspeicher ausgerüstet war.

3.2 Programmiersprachen

Im Rahmen dieser Arbeit wurden für die Teilprojekte MapOmnia sowie BRIME unterschiedliche Programmiersprachen verwendet. Der Pathway-Editor MapOmnia wurde hierbei vollständig in C++ entwickelt, während für das Webprojekt BRIME JavaScript sowie PHP Verwendung fanden.

Programmiersprachen lassen sich in die Kategorie der Interpretersprachen und in die Kategorie der Compilersprachen einteilen. Interpretersprachen benötigen zur Ausführung ein Hilfsprogramm, den Interpreter, welches den ausführbaren Maschinencode erzeugt. Der Interpreter verarbeitet hierbei den Quellcode sequenziell (Internetdokument Programmiersprachen, 2013).

Bei Compilersprachen wird der Quellcode hingegen von einem Compiler übersetzt, wobei ein autonomes Programm entsteht, welches kein weiteres Programm für seine Ausführung benötigt. Interpretierte Programme weisen nachweisbare Geschwindigkeitsnachteile und Performanceverluste auf, da der Interpreter seinerseits Ressourcen belegt. Von Vorteil hingegen ist, dass Interpretersprachen bei Änderungen nicht jedes Mal neu kompiliert werden müssen (Misch).

3.2.1 Programmiersprache C++

Die Programmiersprache C++ wurde von Bjarne Stroustrup entwickelt. Sie gehört zu den kompilierten Programmiersprachen und existiert seit 1983. C++ kann hierbei als Erweiterung von C zu einer objektorientierten Programmiersprache angesehen werden (RRZN, 2011). C++ unterstützt verschiedene Programmierparadigmen, wie die prozedurale, modulare und strukturierte Programmierung. Ebenfalls unterstützt C++ die generische Programmierung durch den Einsatz von Templates und die Definition eigener Datentypen (Stroustrup, 2009).

Bei großen Softwareprojekten, wie dem entwickelten Pathway-Editor MapOmnia, müssen Kriterien wie Robustheit, Korrektheit, Erweiterbarkeit und Wiederverwendbarkeit eingehalten werden. Mit Blick auf genau diese Optimierungen wurde die Programmiersprache C++ entwickelt (RRZN, 2011).

Um die Robustheit der entstehenden Programme zu erhöhen, wird das Programmierparadigma der objektorientierten Programmierung (OOP) angewendet. Die

Grundidee hierbei ist Zusammenfassung von Datenkomponenten und Methoden in Objekten definiert in einer Klasse (RRZN, 2011). Eine Klasse ist eine Datenstruktur in C++ und kapselt zusammengehörige Daten und Funktionen. Ein Objekt ist eine Ausprägung bzw. eine Instanz einer Klasse (Wieland, 2001). Die zentrale Eigenschaft von Objekten, die die Robustheit erhöht, wird durch die Kapselung erhalten. Hierbei werden Kompetenz und Verantwortlichkeit für die Verwaltung von Objektdaten dem Objekt zugeschrieben (RRZN, 2011).

Die Korrektheit bei C++ wird durch strenge Typenkontrolle und Deklarationszwang gegenüber C verbessert (RRZN, 2011).

Durch das Konzept der Vererbung wird die Erweiterbarkeit und Wiederverwendbarkeit von C++ entwickelten Programmen maßgeblich erhöht. Wurde eine Klasse in C++ entwickelt, können dessen Eigenschaften in abgeleiteten Klassen übernommen werden. (RRZN, 2011).

3.2.2 Programmiersprache JavaScript

Die Programmiersprache JavaScript wurde 1995 von der Firma Netscape entwickelt und dient dazu, statische Inhalte von Webseiten dynamisch zu gestalten (Noack, 2011). JavaScript gehört zu den interpretierten Programmiersprachen, welche den objektorientierten Programmieransatz aber auch das funktionale Paradigma verfolgt (Flanagan, 2007). Der Einsatz von JavaScript erfolgt hauptsächlich clientseitig im Webbrowser, kann jedoch auch in einigen Webservern eingesetzt werden (Noack, 2011).

3.2.3 Programmiersprache PHP

Für das Webprojekt BRIME wurde PHP als serverseitige Programmiersprache eingesetzt. PHP wurde von Rasmus Lerdorf 1994 entwickelt und wurde konzipiert, um dynamische Webseiten zu erstellen. Die Programmiersprache zeichnet sich durch Plattformunabhängigkeit aus und steht unter der PHP-Lizenz (Noack, 2010). PHP gehört zu den interpretierten Programmiersprachen und unterstützt die Programmierparadigmen der prozedurale, imperativen, reflektiven und seit PHP4 ebenfalls die objektorientierte Programmierung.

Um dynamische Webseiten mittels PHP zu erstellen, wird es in HTML eingebettet. Dieser Code wird auf dem Server ausgeführt, welcher seinerseits dynamisch HTML-Ausgaben generiert (Internetdokument PHP).

3.3 Auszeichnungssprachen

In den folgenden Unterkapiteln wird kurz auf die Bedeutung der Auszeichnungssprachen XML sowie HTML eingegangen. Im Weiteren werden zwei unterschiedliche Parser-Typen, der DOM und SAX Parser, vorgestellt.

3.3.1 XML - Extensible Markup Language

Da XML (Extensible Markup Language) einigen Dateiformaten aus Kapitel 3.8 als Grundlage dient, wird kurz eine Einführung in die Besonderheiten von XML gegeben. Eine detaillierte Beschreibung hierzu findet sich beispielsweise auf der Webseite http://www.w3schools.com.

XML als Ableitung von SGML (Standard Generalized Markup Language) ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten und wird für den plattform- und implementationsunabhängigen Datenaustausch eingesetzt. Seit 1998 ist XML ein W3C-Standard (World Wide Web Consortium, http://www.w3c.org), wobei die Entwicklung von XML 1996 begann (W3C Communications Team, 2001).

XML ist ein Textformat, besitzt Baumstruktur und benutzt Tags, um Daten abzugrenzen. XML unterliegt strikten Regeln. So wird eine XML-Datei bei weggelassenem Tag oder einem Attribut ohne Anführungszeichen unbrauchbar. Ein Vorteil von XML ist, dass es von Texteditoren lesbar und somit auch neben dem Zielprogramm von weiteren Programmen editierbar und darstellbar ist. Auch ist XML modular: Es erlaubt dem Benutzer, ein neues Dokumentenformat zu definieren (W3C Communications Team, 2001).

Ein XML-Dokument ist wohlgeformt (englisch: well-formed), wenn es grundlegende XML Bedingungen erfüllt. Hierzu zählt, dass sämtliche Elemente geschlossen werden, dass leere Elemente durch einen Slash (deutsch: der Schrägstrich) vor der abschließenden spitzen Klammer gekennzeichnet werden und dass sämtlich Attributwerte in Anführungszeichen eingeschlossen sind (Becker, 2000).

Außerdem ist ein XML-Dokument erst gültig, wenn es mit einer DTD (Document Type Definition) verbunden ist. Der Zweck der DTD besteht darin, zu definieren, welche Elemente es gibt und wie die Struktur des Dokumentes ist (Heymann, 2003).

3.3.2 DOM und SAX

Um XML Dokumente zu lesen und innerhalb eines Programmes zu verarbeiten, wird ein XML Parser benötigt. Grundlegend lassen sich die Parser danach einteilen, ob sie validieren oder nicht und welche Schnittstelle sie zum Zugriff auf das XML-Dokument anbieten (Kunze, 2001). Validierende Parser prüfen das Dokument auf Wohlgeformtheit und Gültigkeit (vergleiche Kapitel 3.3.1).

Der Zugriff auf ein XML-Dokument kann zum einen über ein Document Object Model (DOM) und zum anderen über die Simple API for XML (SAX) erfolgen.

Wird ein XML-Dokument mittels eines SAX Parser verarbeitet, so geschieht dieses eventgesteuert und definierte Rückruffunktionen (englisch: callback function) werden aufgerufen. Dieses gilt sowohl für Ereignisse, wie z. B. dem Auslesen eines Tags, als auch für Fehler oder Warnungen (Zippel, 2008). Daten werden bei diesem sequenziellen Auslesen nicht zwischengespeichert und sind nach der Verarbeitung verloren.

Bei einem DOM-Parser hingegen wird das XML-Dokument als Baumstruktur repräsentiert und im Speicher vorgehalten. Somit sind DOM Parser sehr speicherintensiv, erlauben aber außer dem Lesen von XML-Dokumenten auch die Manipulation des XML Baumes (Harold, 2002).

Der in der Softwaresuite verwendete Parser ist hierbei ein DOM-Parser, der innerhalb des Qt-Frameworks (siehe Kapitel 3.7.2) bereitgestellt wird.

3.3.3 HTML - Hypertext Markup Language

Um Informationen weltweit zugänglich zu gestalten, wurde das World Wide Web *(*Web*)* etabliert. Hierbei ist die benutze Publikationssprache HTML (HyperText Markup Language) (Internetdokument HTML 4.01-Spezifikation). Diese wird von einem Webbrowser dargestellt.

Wie auch bei XML handelt es sich bei HTML um eine Auszeichnungssprache. Diese Auszeichnungssprache, die sich von SGML ableitet, wird vom World Wide Web Consortium (W3C) weiterentwickelt (World Wide Web Consortium, 2013) und beschreibt

die logischen Bestandteile eines Dokumentes. Sie enthält daher Befehle zum Auszeichnen typischer Elemente eines Dokumentes. Hierzu gehören z. B. Überschriften, Textabsätze, Listen oder Tabellen (Noack, 2011). HTML verwendet ebenfalls wie XML Tags und Attribute, jedoch mit dem Unterschied, dass innerhalb von HTML die Bedeutung dieser festgelegt ist (W3C Communications Team, 2001).
HTML ist im Gegensatz zu XML weniger strikt, was zur Folge hat, dass ein weggelassenes Tag oder ein Attribut ohne Anführungszeichen innerhalb eines HTML Dokumentes toleriert und oftmals explizit erlaubt wird (W3C Communications Team, 2001).

3.4 Datenbanksystem

Um große Mengen von Daten effizient zu speichern, werden Datenbanken benutzt. Das Datenbanksystem hierfür besteht aus zwei Teilen: zum einen der Datenbank, in der die Daten gespeichert werden, und zum anderen dem Datenbankmanagementsystem, welches die Daten verwaltet. Zu den Verwaltungsaufgaben gehören unter anderem die strukturierte Speicherung der Daten und der kontrollierte Zugriff auf die Datenbank (Wikipedia, 2013c). Das verbreitetste Datenbankmanagementsystem ist das relationale.
Über Datenbanksprachen ist es dem Benutzer möglich, die gespeicherten Daten zu verwalten und abzufragen. SQL ist hierbei eine genutzte Datenbanksprache, welche das Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen der Daten ermöglicht (Matthiesen und Unterstein, 2003).

3.5 TCP/IP-Protokoll

TCP/IP (Transmission Control Protocol und Internet Protocol) ist eine Protokoll-Kombination aus dem TCP (Transmission Control Protocol) und dem IP (Internet Protocol). TCP ist ein verbindungsorientiertes Protokoll, das für die Datensicherheit und dessen Flusssteuerung sorgt. Hierbei werden die Daten aufgeteilt und mit einem Header versehen, um das Zusammensetzen und die Prüfung zu ermöglichen. Diese Datenpakete werden dann an das Internet Protocol (IP) übergeben (Elektronik-Kompendium).
Das Internet Protokoll hat maßgeblich die Aufgabe Datenpakete zu adressieren und innerhalb des Netzwerkes zu vermitteln. Um eine Identifizierung zu ermöglichen, haben die am Netzwerk teilnehmenden Rechner eine IP-Adresse (W3C Communications Team).

3.6 Programme

Für die Entwicklung des Pathway-Editors MapOmnia sowie des Webprojektes Brime kamen einige Programme zum Einsatz. Zu diesen Programmen gehören die Entwicklungsumgebungen Qt Creator (Kapitel 3.6.1) und Eclipse (Kapitel 3.6.2), das Grafikbearbeitungsprogramm Gimp (Kapitel 3.6.3) und das Dokumentationswerkzeug Doxygen (Kapitel 3.6.4). Der Apache HTTP Server (Kapitel 3.6.5) fand innerhalb des BRIME-Projektes Verwendung. In den folgenden Unterkapiteln werden diese Programme und ihre Features näher beleuchtet.

3.6.1 Qt Creator

Der Pathway-Editor MapOmnia wurde im Qt Creator (Nokia Corporation, 2013b) von Nokia entwickelt. Der Qt Creator ist eine plattformübergreifende Entwicklungsumgebung und läuft auf Windows, Linux/X11 und Mac OS X-Systemen. Er besitzt einen umfangreichen Quellcode-Editor für C++, ein kontextsensitives Hilfesystem, einen visuellen Debugger sowie einen integrierten UI-Designer. Diese Entwicklungsumgebung unterstützt auch die verbreitetsten Versionskontrollsysteme sowie Projekt- und Build-Management-Tools. Die letzte verwendete Version des Qt Creator ist 2.5.2.

3.6.2 Eclipse

Für die Entwicklung des Webprojektes BRIME wurde die integrierte Entwicklungsumgebung Eclipse (Foundation) genutzt. Eclipse ist eine umfangreiche und kostenlose Entwicklungsumgebung, die in Java programmiert ist und für fast alle Systeme verfügbar ist. Da Eclipse sehr viele Programmiersprachen unterstützt, konnte es sowohl für die dynamische Webprogrammierung in PHP als auch für die Programmierung des clientseitige JavaScript-Codes genutzt werden. Der Quellcode Editor bietet Quelltext-Vervollständigung (englisch: Code Completion) sowie die Überprüfung der Syntax (englisch: Syntax Checks). Ebenso verfügt Eclipse über einen Debugger und einer integrierten Dokumentation. Die verwendete Version von Eclipse ist Helios Sr2.

3.6.3 Gimp

Bei der Entwicklung von grafischen Anwendungen ist es wichtig, dem Benutzer über Symbolbilder (englisch: Icons) eine visuelle Schnittstelle zu bieten. So erhalten

Programmfunktionen Wiedererkennungswert und der Umgang mit dem Programm wird erleichtert. Zu diesem Zweck wurde das kostenlose Bildbearbeitungsprogramm Gimp (GNU Image Manipulation Program) (The GIMP Team) genutzt, um Icons sowie Grafiken zu erstellen.
Gimp verfügt über Standardwerkzeuge sowie viele Filterfunktionen. Der Export und Import von zahlreichen Bildformaten wird von Gimp unterstützt. Gimp wurde in der Version 2.8 verwendet.

3.6.4 Doxygen und Doxywizard

Ein sehr hilfreiches Software-Dokumentationswerkzeug für die Erstellung einer übersichtlichen Dokumentation ist Doxygen (van Heesch, 2013) mit seinem grafischem Frontend Doxywizard. Doxygen wurde von Dimitri van Heesch entwickelt und bietet umfangreiche Möglichkeiten Dokumentationsformate zu erstellen. Insbesondere HTML, LaTeX, XML, RTF, PostScript und PDF sind als Exportformate zu erwähnen. Es steht unter der GNU General Public License (Internetdokument Wiki Doxygen).
Doxygen analysiert den Quellcode und erkennt Klassen, Methoden und Variablen. Durch das Erstellen von eigenen Kommentaren werden Beschreibungen zu den jeweiligen Programmelementen erstellt (Internetdokument Wiki Doxygen). Die verwendete Version von Doxygen ist Version 1.8.5

3.6.5 Apache HTTP Server

Der Apache HTTP Server (The Apache Software Foundation, 2013) ist ein frei verfügbares Softwareprojekt, welches von der Apache Software Foundation entwickelt wurde und das Ziel verfolgt, dem Benutzer einen robusten HTTP-Webserver anzubieten (The Apache Software Foundation, 2012). Es bietet die Möglichkeit, durch die Verwendung serverseitiger Skriptsprachen, Webseiten dynamisch zu erstellen. Hierbei werden verschiedene Skriptsprachen wie PHP, Perl oder Ruby direkt, weitere durch das Einbinden von Modulen unterstützt. Die verwendete Softwareversion ist Version 2.0.

3.7 Bibliotheken und Frameworks

Programmbibliotheken und Frameworks bieten dem Programmierer die Möglichkeit, die Entwicklung eigener Software zu beschleunigen. Man kann Programmbibliotheken in die

Kategorien der Quelltextbibliotheken, der statischen Bibliotheken und der dynamischen Bibliotheken unterteilen. Quelltextbibliotheken sind noch nicht kompiliert, statische und dynamische Bibliotheken hingegen schon. Frameworks bieten dem Programmierer ein Programmiergerüst und geben somit die Softwarearchitektur vor. In den folgenden Unterkapiteln werden kurz die Bibliotheken sowie Frameworks charakterisiert, welche Verwendung fanden.

3.7.1 C++-Standardbibliothek

Die C++ Standardbibliothek bietet eine Sammlung von Klassen und Funktionen und wurde maßgeblich von der von Hewlett-Packard entwickelten Standard Template Library beeinflusst. Zu dieser Programmbibliothek gehören unter anderem Container, Iteratoren, Algorithmen, Funktionsobjekte, Zeichenkettenverarbeitung, Numerik sowie Fehlerdiagnose. Viele dieser Komponenten liegen in Form von Templates vor (Kuhlins und Schader, 2005).

3.7.2 Qt

Die Entwicklung des Pathway-Editors MapOmnia beruht im Kern auf der Verwendung des Qt Frameworks (Nokia Corporation, 2013a). Aus diesem Grund wird der Beschreibung von Qt und insbesondere verschiedene Funktionalitäten besondere Beachtung geschenkt.
Qt ist ein Framework, mit dem man plattformübergreifend grafische Benutzeroberflächen (englisch: Graphical User Interface; GUI) erstellen kann. Qt bietet allerdings noch weitere Funktionalität. Es besitzt unter anderem eigene DOM- und SAX-Parser für XML Dokumente (siehe Kapitel 3.3), unterschiedliche Datenbanktreiber für die Anbindung von Datenbanksystemen (siehe Kapitel 3.4), die Unterstützung von Netzwerkoperationen für HTTP und FTP und eine performante Umsetzung für die Visualisierung zweidimensionaler Grafiken. Die verwendete Version des Qt Frameworks ist 4.8.5.
In den folgenden Unterkapiteln wird kurz auf die besonderen Funktionalitäten, die Verwendung in den Projekten fand, eingegangen.

3.7.2.1 Das Qt Property System

Qt hat ein plattformunabhängiges Attribut-System (englisch: Property-System) eingeführt. Hierbei verhält sich eine Property ähnlich einem Klassenmember für Daten. Zusätzlich

sind die Attribute durch das von Qt etablierte Meta-Objekt-System zugänglich. Dieses Meta-Objekt-System ist hierbei für alle Klassen zugänglich, die von der zentralen Klasse *QObject* erben. Das besondere an diesen Attributen ist, dass sie komfortabel abzufragen und zu setzten sind. (Molkentin, 2006). In Quellcode 3.1 ist beispielhaft die Deklaration eines Attribute sowie die dazu gehörenden Klassenmember gezeigt.
Durch diesen Mechanismus ist ein einfaches Setzen von Eigenschaften von erzeugten Objekten möglich. So kann z. B. eine Datenbanktabelle ausgelesen werden und ein schon erzeugter Knoten eines Netzwerkes kann einfach manipuliert und verändert werden, indem seine Eigenschaften überschrieben werden.

Quellcode 3.1: In dem folgenden Quellcodebeispiel wird exemplarisch das *QProperty*-System vorgestellt. Die Klassenmember *highlighted* und *setHighlighted* werden hierbei durch das *Q_Property*-Makro verarbeitet

```
// highlighted
Q_PROPERTY(bool highlighted WRITE setHighlighted READ highlighted)
public:

bool highlighted() const { return m_highlighted;}
void setHighlighted(const bool &highlighted){m_highlighted= highlighted;}
```

3.7.2.2 Undo/Redo-Funktionalität

Qt bietet ein Undo-Framework, welches dem Programmierer erlaubt, die Applikation mit einer Undo/Redo-Funktionalität zu versehen. Die Grundidee hierbei basiert auf dem Kommando-Entwurfsmuster (englisch: Command Design Pattern). Es wird für jede zu verwaltende Änderung eine Instanz eines Kommando-Objekts (englisch: command object) erzeugt. Dieses Objekt seinerseits verwaltet alle relevanten Veränderungen des Dokumentes und ist in der Lage diese Veränderung auszuführen. Durch Speichern dieser Kommando-Objekte auf einem Stapelspeicher (englisch: Stack), können komfortabel Zustandsänderungen des Dokumentes rückgängig bzw. wiederhergestellt werden.

3.7.2.3 Die Model/View Programmierung

Qt setzt für die Visualisierung von Daten die durch Smalltalk bekannte Modell-Präsentation-Steuerung (englisch: Model–View-Controller; MVC) ein. Hierbei kann das

Modell als Datenobjekt, der View als Visualisierung der Daten und der Controller als Interaktions-Interface für den Benutzer gesehen werden.

Qt stellt einen durch den MVC inspirierten Ansatz vor: die Model/View Architektur. In Abbildung 3.1 ist die Architektur schematisch dargestellt. Bei diesem Ansatz wird anstelle der Steuerung ein Delegate (deutsch: der Stellvertreter/der Vermittler) eingesetzt. Hierdurch kann sowohl die Visualisierung der Daten als auch die Interaktion mit den Daten noch weiter verfeinert und selbst definiert werden.

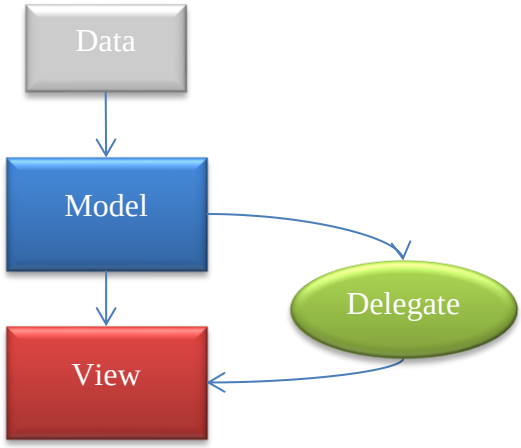

Abbildung 3.1: The Model/View Architektur von Qt. Das Modell beinhalten bzw. verwaltet die Daten. Durch einen View werden dem Benutzer die Daten visualisiert. Das Delegate fungiert hierbei als Vermittler. So können die Visualisierungsmöglichkeiten und die Interaktionsmöglichkeiten mit den Daten angepasst werden.

Qt bietet die Klasse *QAbstractItemDelegate* als abstrakte Basisklasse für ein Delegate an. Durch Implementieren einer eigenen Klasse zur Basis dieser Delegate-Basisklasse kann das Darstellungsverhalten für den jeweiligen Datentyp selbst definiert werden. Ebenso kann bestimmt werden, wie der Benutzer mit den Daten interagieren kann. So kann z. B. einem bestimmten Datentyp ein bestimmter Editor zugewiesen werden.

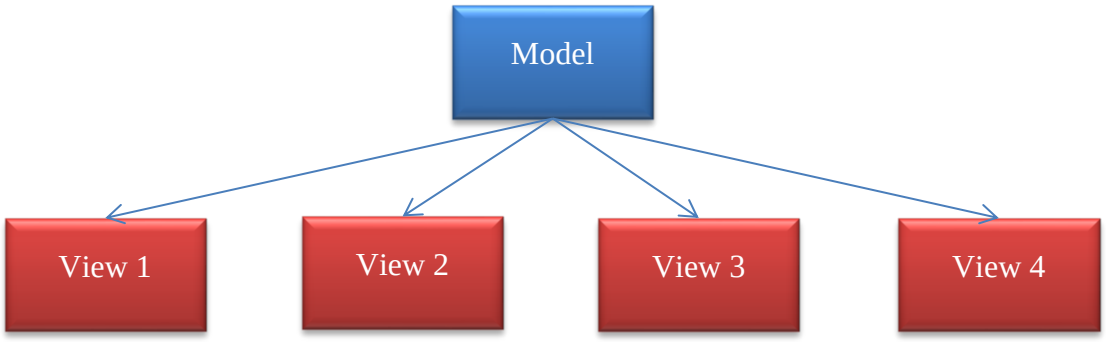

Abbildung 3.2: Ein existierendes Datenmodell kann innerhalb der Model/View Architektur durch beliebig viele Ansichten (englisch: View) dargestellt werden.

Die Model/View Architektur bietet zudem den Vorteil, dass man für ein etabliertes Modell beliebig viele Ansichten (englisch: View) erstellen kann, um das Modell zu visualisieren, ohne die Datengrundlage zu verändern. In Abbildung 3.2 wird dieses schematisch dargestellt.

Die in Qt implementierten Datenmodelle haben die Interface-Klasse *QAbstractItemModel* als Basis. Auch hier kann der Programmierer das Verhalten des Modells selbst definieren, indem er Klassen erstellt, welche von dieser abstrakten Interface-Klasse abgeleitet werden. Allerdings gibt es in Qt schon zahlreich implementierte Modellklassen. Erwähnenswert ist hierbei das *QStandardItemModel*, das einen generischen Ansatz liefert, um Daten unterschiedlicher Datentypen zu speichern. Dieses Datenmodell wird innerhalb des Pathway-Editor-Projekts MapOmnia unter anderem für die Speicherung von Attributen der Netzwerkobjekte benutzt (vergleiche Kapitel 4.1.4.2). Ein Modell wird hierbei aus *QStandardItems* aufgebaut, die eine Zelle innerhalb der Datenmatrix repräsentieren. Jede Zelle ihrerseits hat beliebig viele Ebenen (englisch: Layer), in denen für das dazugehörige Item, Daten vom Typ *QVariant* gespeichert werden können. *QVariant* kapselt hierbei viele unterschiedliche Datentypen. Abbildung 3.3 skizziert dieses Datenmodell schematisch.

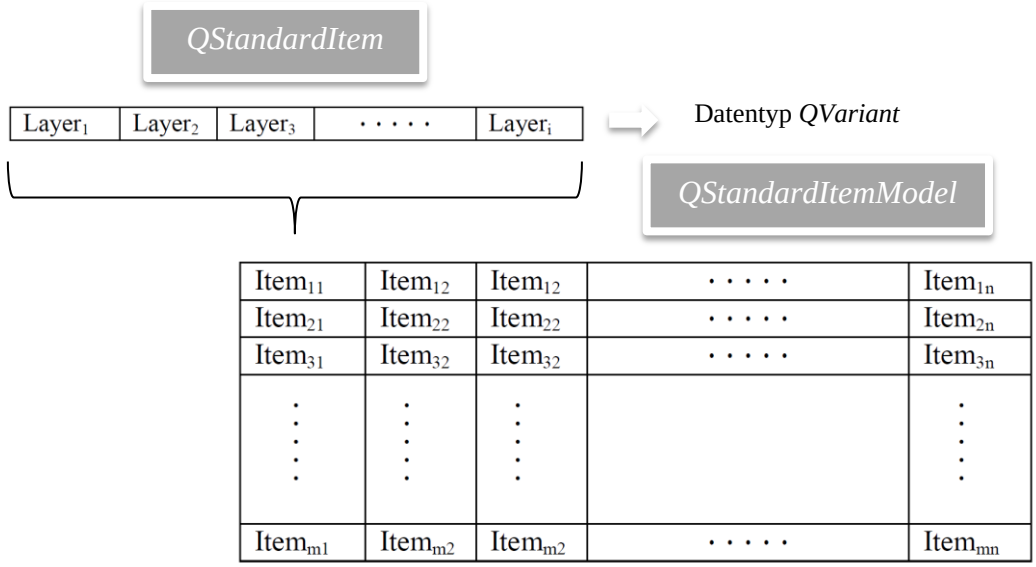

Abbildung 3.3: Die von Qt angebotene Modellklasse *QStandardItemModel* enthält in jeder seiner Zelle eine Instanz eines *QStandardItem*. Dieses speichert seinerseits in Layern Datenwerte, die in dem Datentyp *QVariant* gekapselt sind.

Innerhalb des Model/View Framework bietet Qt eine komfortable Möglichkeit Datenmengen, welche auf Basis des *QAbstractItemModel* implementiert wurden, zu filtern und zu sortieren. Zu diesem Zweck wurde das *QSortFilterProxyModel* von Qt eingeführt. In dieser Klasse kann der Programmierer das Filter- und Sortierverhalten, abhängig vom Datentyp, selbst bestimmen.

3.7.2.4 Skripterstellung

Innerhalb des Qt-Frameworks wird Qt Script zur Verfügung gestellt. Es basiert auf der Grundlage der ECMAScript Skriptsprache. Hierzu gehören z. B. auch JScript von Microsoft sowie JavaScript von Netscape. Durch Verwendung von Qt-Script können neben den in der ECMAScript-Referenz definierten Funktionen ebenfalls die Funktionen und Attribute (englisch: Properties) abgerufen und angesprochen werden, welche Objekten zugrunde liegen, die von *QObject* abgeleitet sind. Zusätzlich müssen die Instanzen der dieser Objekte der Skript-Engine hinzugefügt werden. Für eine detailliertere Erklärung sei an dieser Stelle auf das von Qt bereitgestellte Benutzerhandbuch verwiesen.

3.7.3 QWT - Qt Widgets for Technical Applications

Die QWT-Bibliothek (Rathmann und Wilgen, 2011) beinhaltet GUI-Komponenten und erweitert Qt für den Einsatz bei der Entwicklung von Programmen mit technischem Hintergrund. Es bietet die Möglichkeit, Kurven in zweidimensionalen Diagrammen zu visualisieren. QWT fand in der Version 6.0.0-rc5 Verwendung.

3.7.4 GSL - Gnu Scientific Library

Die GNU Scientific Library (GSL) (Free Software Foundation (FSF)) ist eine Bibliothek, die dem Programmierer eine Vielzahl numerischer Berechnungen zur Verfügung stellt. GSL beinhaltet mathematische Routinen wie z. B. für die Berechnung von Statistik-Funktionen, der numerischen Integration, Differentiation und der schnellen Fourier Transformation. Bei dieser Bibliothek handelt es sich um freie Software und sie steht unter der GNU General Public Lizenz. GSL wurde in der Version 1.13 verwendet.

3.7.5 Boost

Die Boost-Bibliothek (Dawes *et al.*) beinhaltet portable C++ Bibliotheken. Durch das große Funktionsangebot kann Boost für eine Vielzahl von Softwareprojekten genutzt werden (Dawes *et al.*). Hierzu gehören unter anderem die Verarbeitung von Zeichenketten, Container, Iteratoren sowie die Verwendung von Algorithmik auf Graphen über Metaprogrammierung bis hin zur Speicherverwaltung. Die Boost Lizenz ermöglicht sowohl die kommerzielle als auch nicht-kommerzielle Nutzung. Die verwendete Version von der Boost Bibliothek ist Version 1.47.0.

3.7.6 YUI 2 Library - Yahoo User Interface Library

Um interaktive Web-Applikationen zu erstellen, bietet es sich an, die von Yahoo bereitgestellte YUI 2-Bibliothek (Yahoo!) zu nutzen. Hierbei handelt es sich um ein Open-Source (deutsch: quelloffen) JavaScript und CSS Framework. Die YUI Bibliothek ist sehr performant, robust und bietet zahlreiche DOM Hilfsmittel sowie Benutzerschnittstellen. Der Entwickler kann z. B. Bäume und Datentabellen darstellen oder Menüs kreieren (Yahoo!). Die YUI 2-Bibliothek wurde in der Version 2.8.2r1 verwendet.

3.7.7 LiveSearch

Die LiveSearch-Bibliothek (Stocker, 2008) wurde von Christian Stocker entwickelt und bietet die Möglichkeit, Suchergebnisse innerhalb eines Suchfeldes einer Webseite dynamisch anzeigen zu lassen. Für die Umsetzung wird mittels eines XMLHttpRequest eine Abfrage an den Server gestellt. Dieser liefert das Ergebnis. Die Bibliothek steht unter der Apache Software Lizenz 2.0 (Stocker, 2008).

3.7.8 OverLIB

Möchte man auf einer Webseite kleine Informationsfenster für die Visualisierung zusätzlicher Informationen zu einem bestimmten Bereich innerhalb einer Webseite anzeigen lassen, so kann dieses durch die Verwendung der JavaScript-Bibliothek OverLIB (Bosrup, 2009) geschehen. OverLIB wurde programmiert von Erik Bosrup. Die aktuelle und verwendete Version ist 4.21.

3.8 Dateiformate

Für die Visualisierung von Netzwerken stehen zahlreiche Applikationen zur Verfügung, welche meist eigene Dateiformate definieren (Pavlopoulos *et al.*, 2008). Es existieren neben diesen aber auch fest etablierte Standardformate. Diese Standardformate bieten dem Nutzer den Vorteil, erstellte Netzwerke einfach in verschiedenen Programmen darzustellen oder zu bearbeiten, um programmspezifische Vorteile gezielt nutzen zu können.
Gerade XML als Auszeichnungssprache bietet hierbei eine gute Basis, um ein Austauschformat zu definieren. Durch die in Kapitel 3.3.1 beschrieben Eigenschaften, insbesondere der Möglichkeit des plattform- und implementationsunabhängigen Datenaustauschs, haben sich einige Netzwerkdateiformate etabliert, welche auf Grundlage von XML aufgebaut sind. In den folgenden Unterabschnitten werden diese Formate näher beschrieben.

3.8.1 XGMML-eXtensible Graph Markup and Modeling Language

XGMML (eXtensible Graph Markup and Modeling Language) dient als Erweiterung des GML-Formats (siehe Kapitel 3.8.4) unter Berücksichtigung von XML (siehe Kapitel 3.3.1). XGMML wurde als ein universelles Austauschformat für Graphen zwischen

unterschiedlichen Software-Applikationen entwickelt. Es beschreibt Knoten und Kanten des Netzwerkes durch die Verwendung von Tags (Punin).

3.8.2 KGML-KEGG Markup Language

Das KGML (KEGG Markup Language) Dateiformat wurde als Austauschformat für die KEGG PATHWAY Karten (siehe Kapitel 3.9.2) definiert und basiert auf der Grundlage von XML (siehe Kapitel 3.3.1). Die KEGG PATHWAY Karten werden hierbei manuell erstellt und aktualisiert (Kanehisa Laboratories 2, 2010).

3.8.3 SBML-Systems Biology Markup Language

SBML (Systems Biology Markup Language) basiert auf XML (siehe Kapitel 3.3.1) und beschreibt biochemische Netzwerke. Es ist ein weitverbreitetes Format und wird aktuell von über 230 verschiedenen Programmen unterstützt (Internetdokument SBML, 2013).

3.8.4 GML-Graph Modelling Language

Das GML (Graph Modelling Language) Dateiformat dient als Austauschformat sowie der Modellierung von Graphen. Es hat eine leicht zu erlernende Syntax, ist flexibel und erweiterbar. Eine GML-Datei ist hierarchisch aufgebaut und es lassen sich Definitionen von Knoten und Kanten sowie Attributen festlegen (Universität Passau).

3.8.5 CSV-Character Separated Values

CSV heißt im ursprünglichen Sinne Comma-Separated Values (deutsch: Komma separierte Werte). Jedoch werden häufig die Datenstrukturen durch andere Trennzeichen voneinander getrennt. Mit dem CSV-Format lassen sich komfortabel Tabellen speichern und für den Austausch bereitstellen. Die zu verwendende Zeichencodierung ist nicht festgelegt (Louis und Müller, 2007).

3.8.6 MDL Molfile-Molecular Design Limited

Um den Austausch chemischer Strukturen sowie Reaktionen zu ermöglichen, wurde das Molfile von MDL (Molecular Design Limited) entwickelt (MDL Information Systems, 2001). Es handelt sich um eine Textdatei in der in tabellarischer Form Information der chemischen Strukturen gespeichert sind. Es wird hierbei zwischen dem Kopf (englisch:

Headerblock) und dem Rumpf (englisch: Connection Table) der Datei differenziert. Für den strukturellen Aufbau des Moleküls ist der Rumpf entscheidend.

3.8.7 Ini Format-Initialization Format

Das Ini (Initialization) Dateiformat ist eine Textdatei. Sie fungiert als Konfigurationsdatei (Wikipedia, 2013b) und enthält Sektionen. Zu diesen Sektionen werden Wertepaare zugeordnet, über die Attribute adressiert werden können.

3.8.8 Binär-Format

Bei den bisher vorgestellten Dateiformaten handelt es sich ausschließlich um Text-basierte Formate zur Speicherung von Daten. Das Binär-Format wird häufig von Software-Applikationen für die Speicherung benutzt (QuinStreet Inc). Von Vorteil ist, dass sich Binärformate schneller laden und speichern lassen als Textdateien. Ebenso benötigen sie weniger Speicherplatz auf Massenspeichern. Der Austausch zwischen verschiedene Plattformen ist ebenfalls unproblematisch, da die Zielsysteme nicht versuchen die Daten für das jeweilige System zu konvertieren (Wikipedia, 2013a).

3.9 Datenquellen

Die in den folgenden Unterkapiteln beschriebenen Datenquellen sind für die Umsetzung von MapOmnia und BRIME essenziell. Zunächst wird ein Überblick der biologischen Stoffwechselweg-Datenbanken (siehe Kapitel 3.9.1, 3.9.2 und 3.9.3) gegeben, welche Verwendung fanden, um die BRENDA-Maps Stoffwechselkarte (siehe Kapitel 3.9.5) zu etablieren. Das unter 3.9.4 beschriebene Informationssystem PATRIC wurde zwar nicht für die Erstellung der Karte genutzt, wird aber als Import von Teilnetzwerken angeboten. Im Kapitel 3.9.6 wird kurz ein von Melanie Busch in Python entwickeltes Informationstool, beschrieben, das für die webbasierte Reaktionssuche in BRIME Verwendung fand.

3.9.1 BRENDA-Braunschweig Enzyme Database

BRENDA (BRaunschweig ENzyme DAabase) stellt eine umfassende und weltweit wichtige Datenquelle für Informationen zu Enzymen und Stoffwechselwegen dar (Scheer *et al.*, 2011; Internetdokument Enzyme Database – BRENDA, 2013). Die

Enzyminformationen werden in über 40 Daten- und Informationsfeldern angeboten und reichen von der Nomenklatur, katalysierter Reaktionen, ihrer Spezifität für Substrate und Produkte, bis hin zur Enzymstruktur, ihrer zur Isolierung und Aufreinigung und kinetische Parameter (Scheer *et al.*, 2011).

Von zentraler Bedeutung ist die manuelle Kuration der Enzymdaten, wodurch ein hohes Maß an Zuverlässigkeit und Qualität der Daten gewährleistet ist. BRENDA wurde im Jahr 1987 ins Leben gerufen und bietet neben den Enzymdaten auch umfassende Suchmöglichkeiten nach Enzymen und zahlreiche interessante bioinformatische Tools.

Zu diesen Tools gehört z. B. der „Genome Explorer“, mit dessen Hilfe man über 3000 komplette Genome, Plasmide und Organellen nach spezifischen Genen oder Proteinen durchsuchen kann. Auch der „Ontology Explorer“ wird angeboten, welcher einen einfachen Zugang und Navigation zu zahlreichen Ontologien wie z. B. Gewebe-Ontologien, Phenotyp-Ontologien Anatomie-Ontologien oder auch Ontologien chemischer Substanzen ermöglicht. Die „Substructure Search“ kann verwendet werden, um komfortabel Substrukturen zu konstruieren und anschließend nach ihnen zu suchen.

3.9.2 KEGG-Kyoto Encyclopedia of Genes and Genomes

KEGG (Kyoto Encyclopedia of Genes and Genomes) bietet eine Vielzahl von Online-Datenbanken, die Informationen zu Genomen, metabolischen Stoffwechselwegen und Substanzen offerieren (Kanehisa *et al.*, 2008; Kanehisa *et al.*, 2010; Kanehisa Laboratories 3). Bei KEGG PATHWAY (Kanehisa Laboratories 3) handelt es sich um manuell erstellte Stoffwechselkarten. Hierbei werden Informationen zu den molekularen Wechselwirkungen und den Reaktionsnetzwerken bereitgestellt. Neben einer globalen Stoffwechselkarte können Netzwerke z. B. zu metabolischen Stoffwechselwegen, genetischer Information, Prozessierung zellulären Prozessen sowie humane Krankheiten abgerufen werden. (Kanehisa *et al.*, 2008; Kanehisa *et al.*, 2010). Auch bietet KEGG die Möglichkeit, auf die KEGG PATHWAY Karten eigene Daten zu abzubilden, um Einsicht in die biologische Funktion zu erhalten. Hierzu gehören unter anderem molekulare Daten aus dem Bereich der Genomik, Transkriptomik, Proteomik und Metabolomik (Kanehisa *et al.*, 2008; Kanehisa *et al.*, 2010).

3.9.3 MetaCyc-Encyclopedia of Metabolic Pathways

MetaCyc (Caspi *et al.*, 2012; SRI International, 2011) bietet Zugang zu einer Datenbank, welche nicht-redundante, experimentell aufgeklärte, metabolische Stoffwechselwege enthält. Hierbei werden über 1.700 Stoffwechselwege und mehr als 2.000 verschiedene Organismen angeboten. Es wird neben dem primären Metabolismus auch der sekundäre Metabolismus behandelt. Ebenfalls gibt MetaCyc Informationen zu beteiligten Komponenten, Enzymen und Genen. Die Datenintegration erfolgt durch manuelle Kuration (Caspi *et al.*, 2012).

3.9.4 PATRIC - PathoSystems Resource Integration Center

PATRIC (Gillespie *et al.*, 2011; Virginia Bioinformatics Institute) wurde etabliert, um biomedizinische Informationen für pathogene, bakterielle Infektionen bereitzustellen und zu konzentrieren. Es bietet zahlreiche interessante bioinformatische Tools. Mit Hilfe des „Protein Family Sorters“ lassen sich beispielsweise Proteinfamilien innerhalb von Genomen durch eine Heatmap visualisieren. Das „Comparative Pathway Tool“ ermöglicht es, metabolische Stoffwechselwege verschiedener Genome zu vergleichen und durch Verwendung des „Phylogeny Viewers“ lassen sich phylogenetische Verwandtschaftsbeziehungen erkunden. Interessant für diese Arbeit ist, dass über 5000 verschiedene Genome (Stand 26.08.2012) zum Download bereitgestellt werden, aus denen Teilnetzwerke von Stoffwechselkarten, erstellt werden können.

3.9.5 BRENDA-Maps-Generische Stoffwechselkarte

Innerhalb der Doktorarbeit von Susanne Quester wurde die generische Gesamtstoffwechselkarte BRENDA-Maps (Quester und Schomburg, 2011) entwickelt. BRIME dient der Darstellung dieser Stoffwechselkarte. Es handelt sich hierbei um einen bipartiten Graphen (Kapitel 2.1.2.5), welcher als Knotentypen, Enzyme und Substanzen aufweist. Für die Erstellung dieser Karte wurden Informationen über metabolische Stoffwechselwege aus den vorgestellten Datenquellen BRENDA (siehe Kapitel 3.9.1), KEGG (siehe Kapitel 3.9.2) und MetaCyc (siehe Kapitel 3.9.3) zusammengetragen. Auf Grundlage dieser Daten wurde die Karte per Hand erstellt.

In BRENDA-Maps wurden über 120 biochemische Stoffwechselwege und mehr als 1,600 unterschiedliche biochemische Reaktionen eingepflegt. Die Karte beinhaltet somit über 5400 Substanzen (~1,400 verschiedene) und über 1,600 Enzyme (~1,350 verschiedene).

metabolic_pathways

Die „metabolic_pathways“ stellt die relationale Datenbank dar, in welche BRENDA-Maps gespeichert wird. Von zentraler Rolle für die Erstellung eines Netzwerkes sind die Datentabellen „node“ und „edge“. Die Tabelle „node“ hält hierbei die nötigen Informationen, welche Knoten im Netzwerk vorhanden sind und welches Erscheinungsbild sowie Position diese Knoten haben. Die Tabelle „edge“ gibt Aufschluss darüber, wie Knoten innerhalb des Graphen verbunden sind, und bildet das Grundgerüst des Netzwerkes. In Abbildung 3.4 wird das Datenbankschema präsentiert.

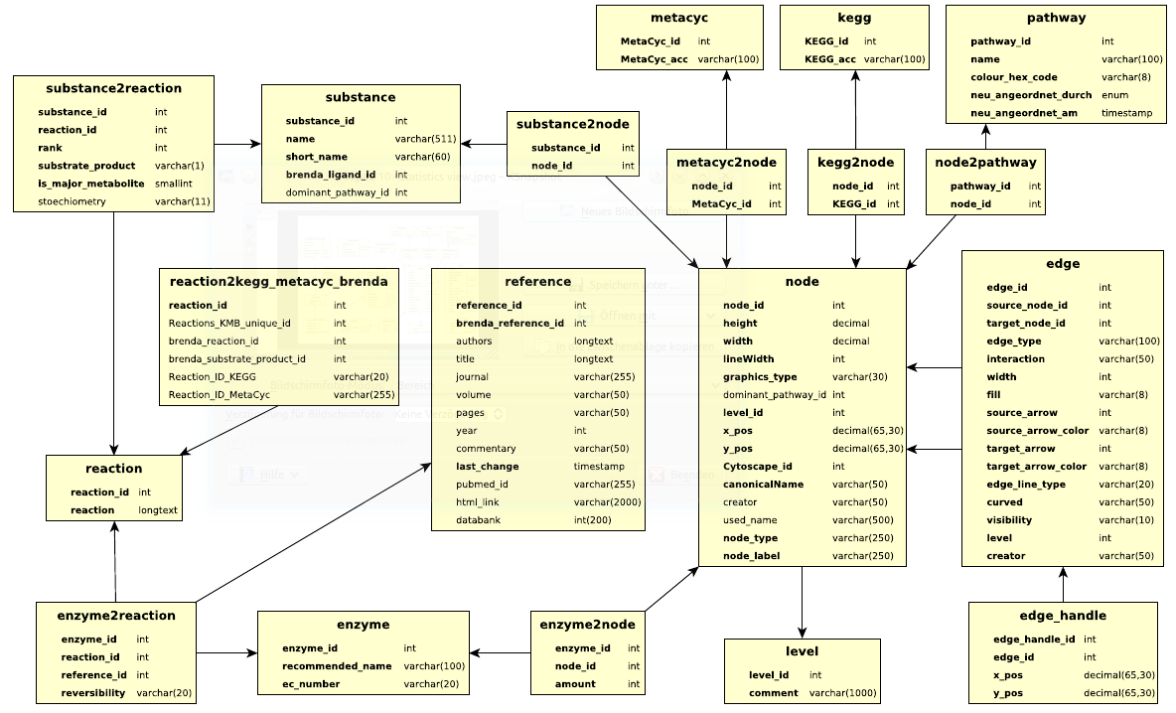

Abbildung 3.4: Datenbanktabellen der Datenbank „metabolic_pathways“ und ihre Beziehungen untereinander (Abbildung aus Busch, 2011).

3.9.6 ReMeRe - Recognition of Metabolites and Reactions

Von Melanie Busch wurde in Python ein Kommandozeilen Tool entwickelt, das in der Lage ist, Komponenten aus BRENDA-Maps zu identifizieren. Das Programm arbeitet auf der Datengrundlage von „metabolic_pathways“, welches in Kapitel 3.9.5 vorgestellt wird. Zu den Suchmöglichkeiten gehören die Suche nach Metaboliten, kompletten Reaktionen, und die Suche nach partiellen Reaktionen. Dieses Tool kann über eine Schnittstelle angesprochen werden und die Suchanfragen auf die generische Stoffwechselkarte ausführen. Von dem Programm werden die gefundenen, eindeutigen Knoten-Ids zurückgegeben.

4 Realisierung und Ergebnisse

In diesem Kapitel wird die Vorgehensweise der Realisierung der beschriebenen Anforderungen aus der Einleitung aufgezeigt. Hierbei wird wiederum zwischen dem Pathway-Editor Projekt MapOmnia und dem Webprojekt BRIME differenziert. Zunächst werden die Ergebnisse von MapOmnia präsentiert (Kapitel 4.1). Danach wird beschreiben, wie der TCP/IP Server konzipiert und implementiert wurde, der als Bindeglied zwischen beiden Hauptprojekten dient (Kapitel 4.2). Anschließend werden die Ergebnisse von BRIME präsentiert (Kapitel 4.3).

4.1 Pathway-Editor MapOmnia

Das Pathway-Editor-Projekt MapOmnia ist eine Softwareapplikation mit vielen unterschiedlichen Funktionalitäten. Im Folgenden werden die Ergebnisse in Unterkapiteln mit unterschiedlichen Schwerpunkten gekapselt. In Kapitel 4.1.1 wird zunächst kurz die Programmmetrik aufgezeigt. Kapitel 4.1.2 beschäftigt sich mit der Quelltext-Dokumentation von MapOmnia durch das Programm Doxygen. Um einen Überblick für das Programm zu bekommen, wird in Kapitel 4.1.3 die Benutzeroberfläche des Pathway-Editors dargestellt. Anschließend werden in Kapitel 4.1.4 elementare Klassen des Programms erläutert, die essenziell für die Funktionsweise der Applikation sind. Kapitel 4.1.5 behandelt die gesamten GUI-Komponenten und die Bedienung von MapOmnia. Hierbei wird insbesondere auf die Leistungsmerkmale und den Funktionsumfang des Programms detailliert eingegangen. Da das Layout von Netzwerken zentrale Funktionalität von MapOmnia darstellt, wird diese im Folgenden beschrieben (Kapitel 4.1.6). Nach dieser theoretischen Betrachtungsweise des Programms folgen Anwendungsbeispiele von ausgewählten Programmeigenschaften (Kapitel 4.1.7). Verwandte Software-Applikationen werden in Kapitel 4.1.8 vorgestellt und anschließend mit MapOmnia verglichen.

4.1.1 Programmmetrik von MapOmnia

Die Anzahl der Quelltext-Zeilen und Anzahl der implementierten Klassen gibt einen Hinweis auf die Komplexität eines Programmes. MapOmnia wurde objektorientiert entwickelt und besitzt über 220 verschiedene Klassen sowie über 3.900 verschiedene Funktionen. Das entspricht im Durchschnitt etwa 18 Funktionen pro Klasse, welches ein

überschaubares Maß für eine Klasse darstellt. Die effektive Anzahl an Quelltextzeilen beträgt ca. 42.500 (zum Vergleich: diese Doktorarbeit besitzt circa. 10.000 Zeilen). Mit durchschnittlich 11 Quelltextzeilen pro Funktion bleibt auch die Komplexität der Funktionen überschaubar. In Tabelle 4.1 sind die einzelnen Metriken gelistet.

Tabelle 4.1: Übersicht der Programmmetrik von MapOmnia

Typ	Anzahl
Dateien	362
Klassen	226
Funktionen	3.957
Zeilen gesamt	70.127
Quelltextzeilen	42.761
Kommentarzeilen	11.845
Leerzeilen	13.268
Deklarative Anweisungen	14.956
Ausführbare Anweisungen	16.132
Verhältnis: Funktionen pro Klasse	18
Verhältnis: Quelltextzeilen pro Funktion	11
Verhältnis: Kommentarzeilen zu Quelltextzeilen	0,28

4.1.2 Quelltext-Dokumentation von MapOmnia

Die Quelltext-Dokumentation des Programmes MapOmnia erfolgt mit dem in Kapitel 3.6.4 beschriebenen Programm Doxygen. Die Dokumentation gibt eine strukturierte Übersicht über Klassen und Funktionen des Programms. In Abbildung 4.1 ist der Klassenindex der erstellten HTML-Dokumentation dargestellt. Durch die vollständige und übersichtliche Dokumentation sollte es anderen Programmierern leichter möglich sein, sich in der komplexen Programmstruktur von MapOmnia zurechtzufinden, um beispielsweise das Programm weiter zu entwickeln.

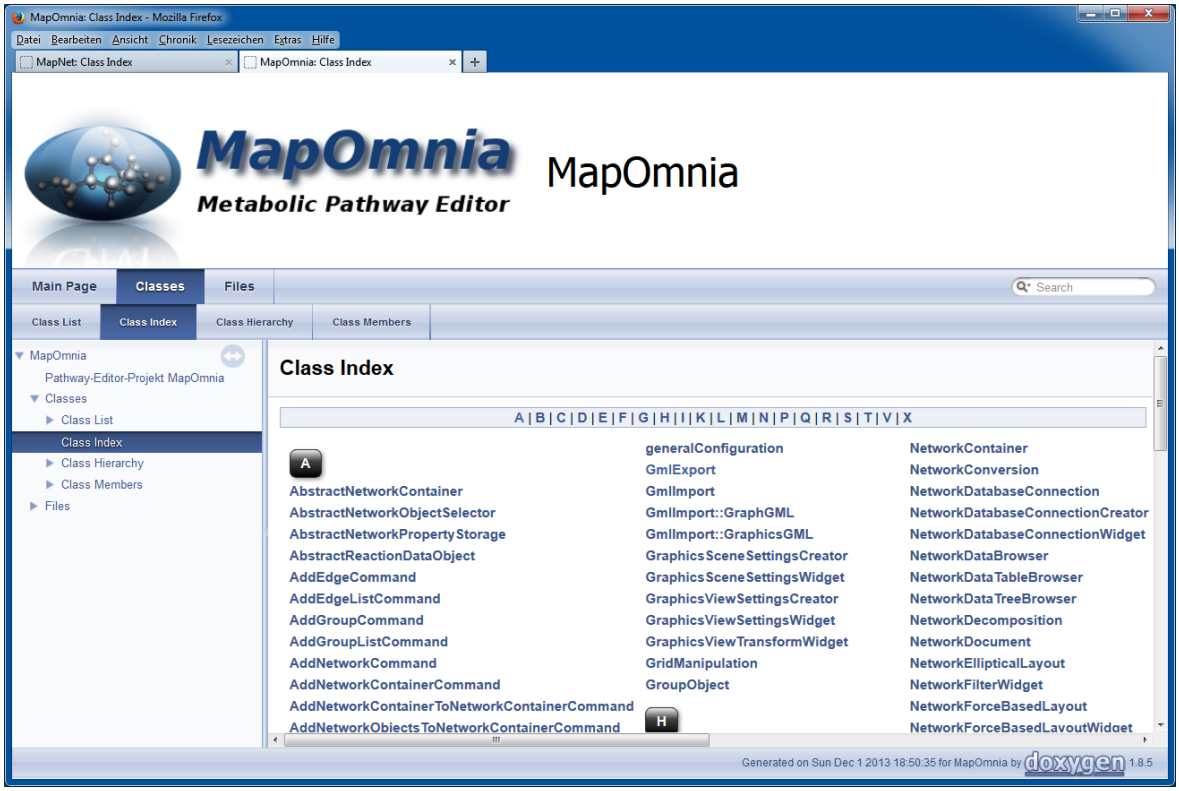

Abbildung 4.1: Die Doxygen-Dokumentation bietet eine strukturierte Übersicht über das entwickelte Programm MapOmnia. Die obige Ansicht zeigt den Klassenindex. Hier sind sämtliche Klassen alphabetisch angeordnet und zu den jeweiligen Klassenbeschreibungen verlinkt. Die vollständige Doxygen-Dokumentation befindet sich auf der zu dieser Arbeit beigefügten Daten-DVD-ROM.

4.1.3 Benutzeroberfläche von MapOmnia

Die Benutzeroberfläche (Abbildung 4.2) besteht im Wesentlichen aus einem Multiple-Document-Interface (MDI, Abbildung 4.2, a), in welchem die geöffneten Netzwerke in Fenstern erstellt werden. Über das Menü (Kapitel 4.1.5.15) in der Menüleiste „Window“ lassen sich diese Fenster verwalten.

In der oberen Menüleiste befinden sich sämtliche Funktionen des Programms, wobei viele ebenfalls als Docking-Fenster aufrufbar sind. Diese können durch Rechtsklick auf die Menüleiste aktiviert sowie deaktiviert werden. Zu diesen Docking-Fenstern gehört der „Undo-Stack Browser“ (Kapitel 4.1.5.3), welcher als grafisches Interface für die Undo/Redo-Funktionalität des Programmes fungiert, der „Network Data Browser“ (Kapitel

4.1.5.4, Abbildung 4.2, b), welcher eine Übersicht über geöffnete Netzwerke liefert sowie das „Network Overview Fenster (Kapitel 4.1.5.11), welches eine miniaturisierte Übersichtskarte anzeigt. Eine detaillierte Beschreibung findet sich in dem Kapitel Komponenten und Bedienung von MapOmnia 4.1.5.

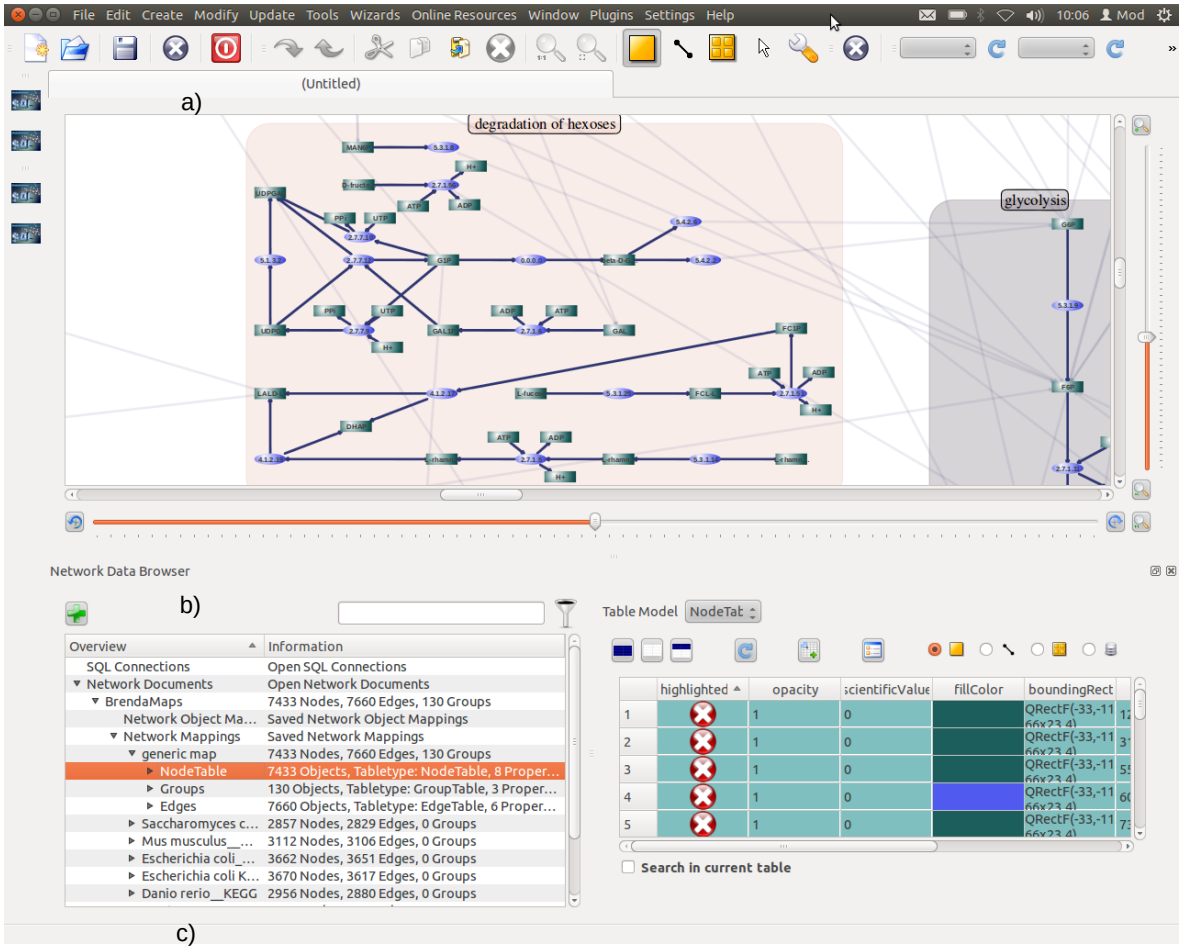

Abbildung 4.2: Benutzeroberfläche (GUI) des Pathway-Editors MapOmnia. Im oberen Bereich befindet sich das Multiple-Document-Interface (a), im unteren befindet sich der „Network Data Browser“ (b). Unterhalb der Menüleiste befinden sich die Toolbars File, Edit Windows und Styles. Links ist die Toolbar für die Auswahl der Funktionen, welche durch Plug-Ins geladen werden, dargestellt.

Das Programm bietet dem Benutzer die Möglichkeit, viele der Funktionen, die in der Menüleiste lokalisiert sind, als Symbolleiste (englisch: Toolbars) anzeigen zu lassen. Es

sind für Basisfunktionalitäten Symbolbilder (englisch: Icons) hinterlegt, die in diesen Symbolleisten angezeigt werden.
In der unteren Leiste (Abbildung 4.2, c) werden dem Benutzer Zustands-Informationen eingeblendet. So wird zum Beispiel das erfolgreiche Laden, Speichern oder auch das Erstellen neuer Netzwerkkomponenten ersichtlich.
Alle Toolbars und Docking-Fenster lassen sich durch „Drag and Drop“ frei positionieren, sodass der Benutzer das Erscheinungsbild individuell anpassen kann. Diese Einstellungen werden beim Verlassen des Programms gespeichert und automatisch beim Neustart aufgerufen.

4.1.4 Klassen von MapOmnia

In diesem Kapitel wird in eine kurze Beschreibung der Klassen gegeben, welche für das Programm MapOmnia von zentraler Bedeutung sind. Zunächst werden die Klassen der Netzwerkobjekte (Kapitel 4.1.4.1) näher erörtert. Hieran schließt sich eine Beschreibung der Datenstruktur eines Netzwerkes an (Kapitel 4.1.4.2). In Kapitel 4.1.4.3 wird auf die aus Kapitel 3.7.2 vorgestellte Delegate-Funktionalität eingegangen. Das Kapitel 4.1.4.4 behandelt das Sortier- und Filter- Proxy Modell, und die Undo/Redo-Funktionalität wird in Kapitel 4.1.4.5 erläutert. Klassen, die für Datenakquise zuständig sind, werden in Kapitel 4.1.4.6 beschrieben. Die Import- und Export-Funktionen des Programms werden in Kapitel 4.1.4.7 erläutert.

4.1.4.1 Netzwerkobjekte

Basisklasse *NetworkGraphicsWidget*

Das *NetworkGraphicsWidget* ist die Basisklasse aller implementierten Netzwerkobjekte. Diese Klasse ist abgeleitet von der Klasse *QGraphicsWidget* und dient dazu in zweidimensionalen Szenen von Qt gerendert dargestellt zu werden. Zu den Netzwerkobjekten gehört das *NodeObject*, das die Knoten eines Netzwerkes repräsentiert, das *EdgeObject*, das die Kanten darstellt und das *GroupObject*, das als Art Superknoten fungiert und selber Knoten und Kanten enthalten kann. Abbildung 4.3 zeigt das Vererbungsdiagramm für die Klasse *NetworkGraphicsWidget*.

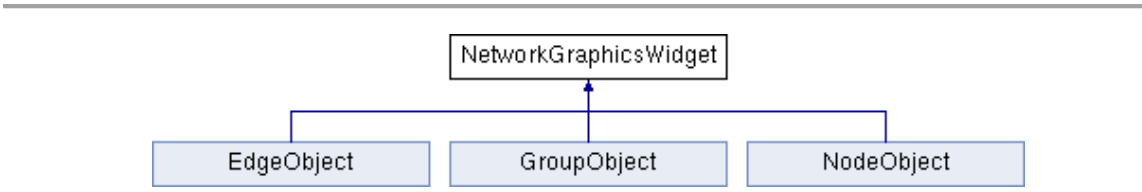

Abbildung 4.3: Vererbungsdiagramm für das *NetworkGraphicsWidget*. *NodeObject*, *EdgeObject*, und *GroupObject* werden von *NetworkGraphicsWidget* abgeleitet und haben daher zahlreiche Methoden gemeinsam.

Netzwerkobjekt *NodeObject*

Die Klasse *NodeObject* dient der grafischen Repräsentation eines Knoten innerhalb eines Netzwerkes. Es besitzt zahlreiche Methoden, um sein Darstellungsverhalten beeinflussen zu können. Darüber hinaus werden Methoden angeboten, die es ermöglichen Knoteneigenschaften wie z. B. den Knotengrad (Kapitel 2.1.2.1) zu ermitteln.

Netzwerkobjekt *EdgeObject*

Die Klasse *EdgeObject* dient der grafischen Repräsentation einer Kante innerhalb eines Netzwerkes. Ebenso wie das *NodeObject* hat sie zahlreiche Methoden, um das Darstellungsverhalten von Instanzen der Klasse beeinflussen zu können. Es kann beispielweise die Strichstärke, Farbe oder Transparenz manipuliert werden.

Netzwerkobjekt *GroupObject*

Die Klasse *GroupObject* dient der grafischen Repräsentation von Gruppierungen von Knoten und Kanten. Die Zusammengehörigkeit wird durch Unterlegung mit derselben Farbe dargestellt. Diese Art der Darstellung kann aber vom Nutzer angepasst werden. So ist es möglich die gesamte Fläche, die von den Knoten und Kanten bestimmt wird, mit einem abgerundeten Rechteck zu unterlegen, oder als konvexe Hülle zu rendern. Da die Knoten als Liste innerhalb eines Objektes des *GroupObjects* gespeichert werden, kann ebenfalls ein farblich unterlegter Weg gerendert werden.

Attribute der Netzwerkobjekte

Netzwerkobjekte haben zahlreiche Attribute (wie Größe, Farbe und Position), die verändert und gespeichert werden können. Neben den fest implementierten Attributen können auch

weitere dynamisch zu erzeugten Objekten hinzugefügt werden. Letztere können ebenfalls in den erstellten Teilnetzwerken (vergleiche Kapitel 4.1.4.2) gespeichert werden, wodurch ein dynamisches Datenmodell nötig ist (vergleiche Kapitel 3.7.2.3). Das Vererbungsdiagramm in Abbildung 4.3 zeigt, dass sich die drei Netzwerkobjekttypen von *NetworkGraphicsWidget* ableiten. Das *NetworkGraphicsWidget* bietet Attribute, welche allen Netzwerkobjekten gemeinsam sind. Eine Übersicht dieser mit kurzer Beschreibung befindet sich in Tabelle A.2 im Anhang.

4.1.4.2 Datenstruktur des Netzwerkes

Das Netzwerk ist definiert durch Knoten und Kanten. Die Kanten verbinden die Knoten und haben daher jeweils einen Start- und Endknoten. Ein weiteres Objekt, das *GroupObject***,** fungiert hierbei als Superknoten, welcher selbst Knoten und Kanten beinhalten kann. Sämtliche Netzwerkobjekte (Knoten, Kanten und Gruppen) besitzen Attribute und ihre Attributwerte bestimmen ihre grafische Repräsentation bzw. ihre Datenwerte für die jeweiligen Objekte.

Teilnetzwerke haben dabei eine Untermenge an Knoten, Kanten und Gruppen (vergleiche Kapitel 2.1.2.4). Innerhalb von Teilnetzwerken können die Netzwerkobjekte andere Attributwerte für manche oder alle Attribute zugeordnet haben. Da es sich bei MapOmnia um einen grafischen Pathway-Editor handelt, mit welchem der Benutzer interagiert, wird für die Speicherung der Attribute und den dazugehörigen Datenwerten das *QAbstractItemModel* von Qt verwendet (siehe Kapitel 3.7.2.3). Der Vorteil ist, dass es sich um ein dynamisches Datenmodell handelt und theoretisch eine unbegrenzte Anzahl verschiedener Datentypen beinhalten kann. Außerdem bietet Qt mit dem Model/View Konzept, das auf dem MVC-Pattern basiert (vergleiche Kapitel 3.7.2.3), direkt die Möglichkeit, die Datenwerte in Ansichten zu visualisieren und zu bearbeiten.

Im Folgenden wird ein kurzer Überblick über die Klassen gegeben, die ein Netzwerk innerhalb von MapOmnia verwalten, wobei für detaillierte Informationen auf die Doxygen Dokumentation (befindlich auf der Daten-DVD) verwiesen sei.

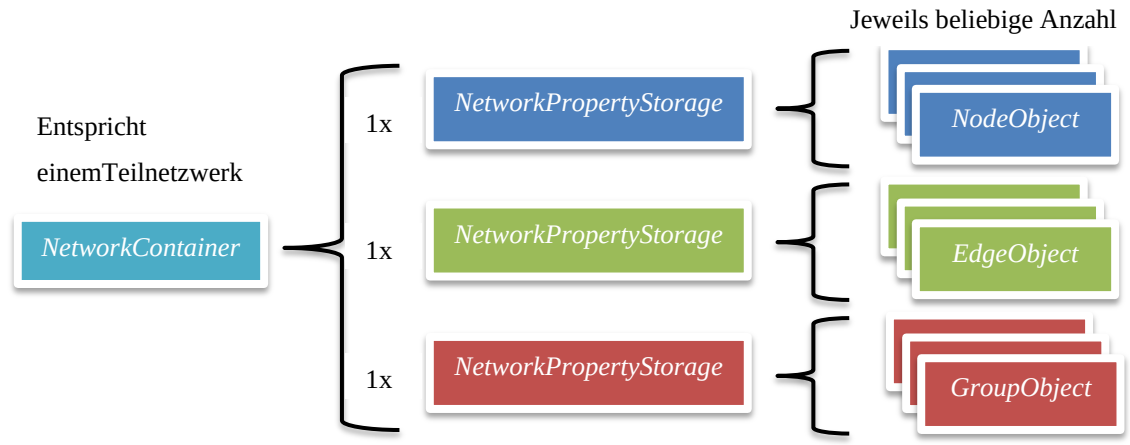

Abbildung 4.4: Schematisch skizzierter Aufbau einer Instanz eines *NetworkContainers*: Ein Teilnetzwerk beinhaltet drei Instanzen der Klasse *NetworkPropertyStorage*; jeweils für Knoten, Kanten und Gruppen des Netzwerks.

Klasse *NetworkPropertyStorage*

Ein Objekt der Klasse *NetworkPropertyStorage* kümmert sich um die gesamte Datenhaltung der Attribute für die Netzwerkobjekte und speichert Pointer ihm zugewiesener Objekte. Prinzipiell kann ein Objekt der Klasse *NetworkPropertyStorage* als Datenhaltung für beliebige andere Objekte dienen, die von *QObject* abgeleitet sind. Es können die Attribute als Liste vorgegeben werden, welche für die Objekte zu speichern sind. Durch Verwendung des Qt Property Systems (vergleiche Kapitel 3.7.2.1) können die Objekte nach diesen Attributen abgefragt werden und der zurückgelieferte Datentyp *QVariant* wird in einem *QStandardItemModel* (vergleiche Kapitel 3.7.2.3) gespeichert. In jeder Zelle des Modells werden für die oberste Zellebene (Display Layer) die Daten gespeichert, die für das Attribut gesetzt wurden. In einer weiteren Zellebene wird der Pointer auf das jeweilige Objekt und in einer dritten das abgefragte Attribut gespeichert. Wenn eine Aktualisierung (Update) der Objekte gewünscht ist, beispielsweise beim Wechsel zu einem anderen Teilnetzwerk, werden die Objekte einfach mit dem Attribut und dem dazugehörigen Datenwert überschrieben.

Klasse *NetworkContainer*

Ein Objekt der Klasse *NetworkContaine*r repräsentiert ein Teilnetzwerk und kümmert sich um die gesamte Datenverwaltung der Attribute sowie Knoten, Kanten und Gruppen eines erstellten Teilnetzwerks. Im Wesentlichen beinhaltet er drei Objekte der Klasse *NetworkPropertyStorage*. Jedes davon dient der Datenhaltung für eines der drei Netzwerkobjekte: Knoten, Kanten sowie Gruppen. Abbildung 4.4 skizziert schematisch diesen Aufbau.

Klasse *NetworkObject*

Innerhalb eines Objekts der Klasse *NetworkObject* werden die erstellten Teilnetzwerke (*NetworkContainer*) gespeichert und verwaltet. Es beinhaltet ebenfalls Erstellungs-Methoden (Factory-Methoden), um Knoten, Kanten, Gruppen und Teilnetzwerke zu erzeugen. Des Weiteren beinhaltet ein Objekt der Klasse *NetworkObject* ein Gesamtnetzwerk, welches alle erstellten Netzwerkobjekte zusammenfasst sowie ein Referenznetzwerk, welches ebenfalls alle erstellten Netzwerkobjekte zusammenfasst diesen jedoch eine definierte Transparenz zuordnet. In Abbildung 4.5 ist dieser Aufbau schematisch skizziert.

Klasse *NetworkDocument*

Die Klasse *NetworkDocument* ist die grundlegendste Klasse für die Verwaltung eines Netzwerks von MapOmnia (vergleiche Abbildung 4.5). Die erzeugten Objekte der Klasse *NetworkDocument* werden von der MDI verwaltet. Ein Objekt von *NetworkDocument* verwaltet hierbei seinerseits ein Objekt der Klasse *NetworkObject*, welches die zentrale Klasse für die Speicherung des Gesamtnetzwerkes und seiner Teilnetzwerke darstellt. Unter anderen erstellt jedes Objekt der Klasse *NetworkDocument* ein Objekt der Klasse *NetworkGraphicsScene* (siehe Kapitel 4.1.5.1), ein *UndoStack* (vergleiche Kapitel 3.7.2.2), das *GraphicsViewTransformWidget* (siehe Kapitel 4.1.5.2) sowie die Skript-Engine (vergleiche Kapitel 3.7.2.4). Die Klasse *NetworkDocument* beherbergt ebenfalls die Undo-basierte Editierfunktionen des Netzwerkes und der Netzwerkobjekte sowie zahlreiche Methoden, welche für den Import sowie Export nötig sind.

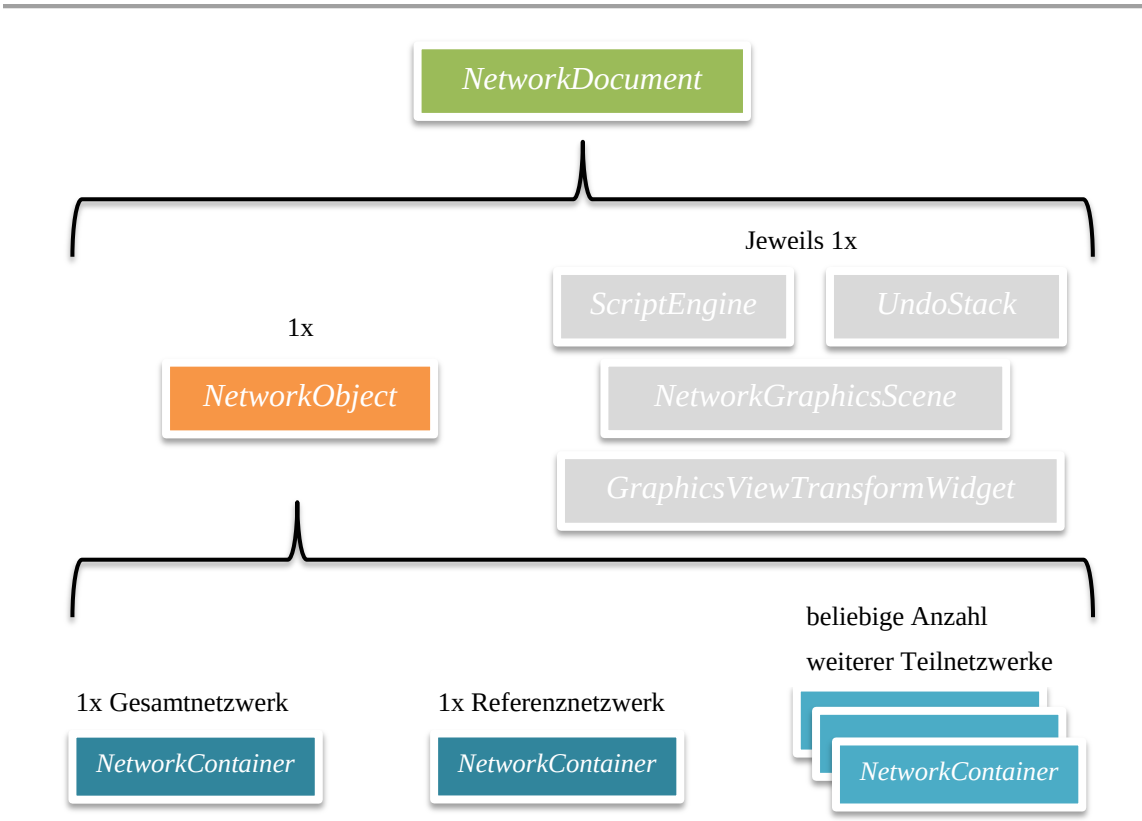

Abbildung 4.5: Schematisch skizzierter Aufbau einer Instanz eines *NetworkDocument* und dem dazugehörigen *NetworkObject*: Das *NetworkObject* beinhaltet mindestens 2 *NetworkContainer*; einmal ein Gesamtnetzwerk, welches sämtliche Knoten, Kanten und Gruppen aller Teilnetzwerke aggregiert, sowie ein Referenznetzwerk, welches ebenfalls diese Menge an Netzwerkobjekten besitzt, jedoch die diesen eine vorgegeben Transparenz zuordnet. Die Anzahl an Teilnetzwerken ist beliebig. Einer Instanz der Klasse *NetworkDocument* ist genau ein *NetworkObject* zugeordnet.

4.1.4.3 Delegate Programmierung

Das verwendete Datenmodell (vergleiche Kapitel 3.7.2.3) hat als grundlegenden Datentyp *QVariant*. Dieser kann seinerseits andere Datentypen kapseln. Die unterstützten Datentypen sind in Tabelle A.1 im Anhang gelistet.

Klasse *PropertyItemDelegate*

Qt bietet die Möglichkeit, Daten datentypabhängig darstellen zu lassen. Zu diesem Zweck wurde die Klasse *PropertyItemDelegate* als Delegate (Kapitel 3.7.2.3) implementiert.

Neben dem Darstellungsverhalten wird von dem Delegate ebenfalls bestimmt, welcher Editor für welchen Datentyp zum Editieren der Daten benutzt werden soll.
Um innerhalb von MapOmnia einen komfortablen Umgang mit Daten zu gewährleisten, ist es nötig, das Darstellungsverhalten von verschiedenen Datentypen anzupassen. Tabelle 4.2 zeigt, für welche Datentypen es angepasst wurde. Hierbei wurde eine besondere Erkennung für Enumeratoren implementiert. Da sich sämtliche Netzwerkobjekte von *QObject* ableiten, lässt sich durch ein MetaObject herausfinden, ob es sich um den momentanen Datentyp, um einen Integer-Wert oder einen Enumerator handelt. Wird erkannt, dass es ein Enumerator ist, wird automatisch der Enumerator-Name für die Darstellung verwendet und nicht der Integer-Wert.
Der Nutzer kann die Datenwerte von Netzwerkobjekten editierten. Hierzu kann er innerhalb der Darstellung der Tabelle mit den Datenwerten (Abbildung 4.2 b-rechts) in eine Zelle der Matrix mit Doppelklick klicken, wodurch der Editiermodus gestartet wird. Für diesen Editiermodus wurden in MapOmnia programmspezifische Editierlösungen implementiert. Tabelle 4.3 gibt hierzu eine Übersicht. Das Erscheinungsbild der meisten Editoren ist in Kapitel 4.1.5 beschrieben. Auch hier findet bei Enumeratoren eine spezielle Bearbeitung statt. Bei einem Enumerator wird ein dynamisches Dialogfenster erstellt, welches die Auswahl für den Enumeratortyp definiert. Dieses ist in Kapitel 4.1.5.16 exemplarisch beschrieben.

Tabelle 4.2: Übersicht über die Datentypen, für die ein individuelles Darstellungsverhalten implementiert wurde

Datentyp	**Beschreibung des Darstellungsverhaltens**
QPoint, QPointF	Punktkoordinate im zweidimensionalen Raum: Es wird der x- und y-Wert angezeigt.
QSize, QSizeF	Größe im zweidimensionalen Raum:. Die Größe besteht aus Höhe und Breite, welche angezeigt werden.
QColor	Farbe: Hierbei wird die Zelle der Tabelle mit der Farbe ausgefüllt dargestellt.
QDate	Datumsstempel: Es wird das Datum angezeigt.
QDateTime	Datumsstempel mit Uhrzeit: Es wird das Datum und die Uhrzeit angezeigt.
QTime	Uhrzeit: Es wird die Uhrzeit angezeigt.
QPen	Der aktuelle Stift: Hierfür wird in die Zelle eine Ellipse gezeichnet, welche den Stift für die Linie wiedergibt.
QBrush	Der aktuelle Pinsel: Hierfür wird in der Zelle eine Ellipse mit dem Pinsel ausgefüllt, dargestellt.
QRect, QRectF	Rechteck: Für ein Rechteck wird in der Zelle die Punktkoordinate im zweidimensionalen Raum für den oberen linken Rechteckspunkt sowie die Höhe und Breite angezeigt.
QStringList	Eine Liste mit Strings: Die Strings, die die Liste enthält, werden kommasepariert in der Zelle dargestellt.
Bool	Wahrheitswert: Der Wahrheitswert wird in Form eines kleinen grünen Häkchens für wahr und einem roten x für unwahr dargestellt.
QFont	Schriftart: Die Schriftart wird namentlich genannt.
QBitmap, QPixmap, QPicture, QImage, QIcon	Bild-Datentyp: Alle Bild-Datentypen werden in der Zelle als Symbolbild angezeigt.

Tabelle 4.3: Übersicht der Zuordnung von Datentypen zu dem jeweiligen Editor, welcher für die Bearbeitung vorgesehen ist

Datentyp	**Implementierter Editor**
QPoint, QPointF	Für die spezielle Bearbeitung für Punktkoordinate im zweidimensionalen Raum wurde der „PointEditor“ implementiert.
QSize, QSizeF	Für die spezielle Bearbeitung für Größe im zweidimensionalen Raum wurde der „SizeEditor“ implementiert.
QColor	Der Editor „ColorEditor“ erlaubt eine Auswahl einer Farbe.
QDate	Für die Bearbeitung eines Datums wird das von Qt bereitgestellte *QDateEdit* benutzt.
QDateTime	Für die Bearbeitung eines Datums wird das von Qt bereitgestellte *QDateTimeEdit* verwendet.
QTime	Für die Bearbeitung einer Uhrzeit wird das von Qt bereitgestellte *QTimeEdit* benutzt.
QPen	Die Einstellung des Stiftes erfolgt durch den Editor „PenDialog“.
QBrush	Die Einstellung des Pinsels erfolgt durch den Editor „BrushDialog“.
QRect, QRectF	Um ein Rechteck zu bearbeiten, wird der Editor „RectangleEditor“ verwendet.
QStringList	Für die spezielle Bearbeitung einer Liste mit Strings wurde der Editor „ListItemSelection“ implementiert.
Bool	Für die Bearbeitung von Wahrheitswerten wird die von Qt bereitgestellte *QCheckBox* benutzt.
QFont	Für die Bearbeitung einer Schriftart wird der von Qt bereitgestellte *QFontDialog* benutzt.
QBitmap, QPixmap, QPicture, QImage, QIcon	Innerhalb von MapOmnia lassen sich Bilder nicht bearbeiten. Die Darstellung erfolgt durch die Verwendung des von Qt bereitgestellten *QLabel*. Aus MapOmnia heraus kann für die Bearbeitung ein Bildbearbeitungsprogramm wie z. B. Gimp geöffnet werden.

4.1.4.4 Sortier- und Filter- Proxy Modell

Klasse *NetworkSortFilterProxyModel*

Durch das von Qt etablierte Model/View Konzept wird ebenfalls eine Sortier- und Filtermöglichkeit angeboten (siehe Kapitel 3.7.2.3). In MapOmnia wurde eine von *QSortFilterProxyModel* abgeleitete Klasse *NetworkSortFilterProxyModel* erstellt. Es muss hierbei für die verschiedenen Datentypen definiert werden, was der Vergleichsoperator „<“ zurückgeben soll. In Quellcode 4.1 wird dieses exemplarisch gezeigt. Im Kapitel 4.1.5.6 wird das Sortier- und Filterverhalten näher beschrieben.

Quellcode 4.1: Das Sortier- und Filterverhalten für einen Datentyp hängt von dem Vergleichsoperator „<“ ab. Der folgende Quellcode zeigt dieses exemplarisch für die Datentypen ByteArray, Char, Color, Date sowie DateTime

```
case QVariant::ByteArray:
    return value1.toByteArray()<value2.toByteArray();
    break;
case QVariant::Char:
    return value1.toChar()<value2.toChar();
    break;
case QVariant::Color:
    return value1.value<QColor>().rgb()<value2.value<QColor>().rgb();
    break;
case QVariant::Date:
    return value1.toDate()<value2.toDate();
    break;
case QVariant::DateTime:
    return value1.toDateTime()<value2.toDateTime();
    break;
```

4.1.4.5 Undo/Redo-Funktionalität

Jedes Programm, das für die Bearbeitung von Daten programmiert ist, gewinnt deutlich an Benutzerfreundlichkeit, wenn eine Möglichkeit besteht, Veränderungen rückgängig zu machen bzw. wieder herstellen zu können. Zu diesem Zweck wurde auch für den Pathway-Editor MapOmnia eine solche Funktionalität implementiert. Wie in Kapitel 3.7.2.2 beschrieben, werden Veränderungen innerhalb von Objekten gekapselt. Diese gekapselten Objekte lassen sich auf einen Stack packen, um sie zu speichern. Für das Programm sind die unter Tabelle 4.4 aufgelistete Kommandos vorgesehen. Da MapOmnia das von Qt vorgesehene Property-System (Kapitel 3.7.2.1) für die Datenmanipulation verwendet, werden sämtliche Veränderungen die für die Netzwerkobjekte möglich sind unter dem

SetObjectProperty Kommando gekapselt, wodurch eine große Vielseitigkeit für die Undo/Redo-Funktionalität besteht.

Tabelle 4.4: Übersicht der implementierten Kommandos für die Undo/Redo-Funktionalität

Kommando	**Bezieht sich auf:**	**Beschreibung**
AddNode	*NetworkContainer*	Fügt dem aktuell ausgewählten Teilgraphen einen neuen Knoten hinzu.
AddEdge	*NetworkContainer*	Fügt dem aktuell ausgewählten Teilgraphen eine neue Kante hinzu.
AddGroup	*NetworkContainer*	Fügt dem aktuell ausgewählten Teilgraphen eine neue Gruppe hinzu.
RemoveNode	*NetworkContainer*	Entfernt aus dem aktuell ausgewählten Teilgraphen den gewählten Knoten.
RemoveEdge	*NetworkContainer*	Entfernt aus dem aktuell ausgewählten Teilgraphen die gewählte Kante.
RemoveGroup	*NetworkContainer*	Entfernt aus dem aktuell ausgewählten Teilgraphen die gewählte Gruppe.
AddNetworkContainer-ToNetworkContainer	*NetworkContainer*	Fügt dem aktuell ausgewählten Teilgraphen alle Knoten, Kanten und Gruppen aus dem zweiten Teilgraphen hinzu.
RemoveNetworkContainer-FromNetworkContainer	*NetworkContainer*	Entfernt aus dem aktuell ausgewählten Teilgraphen alle Knoten, Kanten und Gruppen aus dem zweiten Teilgraphen.
AddNetworkObjects-ToNetworkContainer	*NetworkContainer*	Fügt dem aktuell ausgewählten Teilgraphen alle übergebenen

Kommando	Bezieht sich auf:	Beschreibung
		Netzwerkobjekte hinzu.
RemoveNetworkObjects-FromNetworkContainer	*NetworkContainer*	Entfernt aus dem aktuell ausgewählten Teilgraphen alle übergebenen Netzwerkobjekte.
SetObjectProperty	*NetworkContainer*	Setzt für das übergebene Netzwerkobjekt und dem übergebenen Attribut den übergebenen Wert.
SetObjectsProperty	*NetworkContainer*	Setzt für die übergebenen Netzwerkobjekte, und dem übergebenen Attribut den übergebenen Wert.
AddNetworkContainer	*NetworkDocument*	Fügt zu dem Netzwerkdokument einen neuen Teilgraphen hinzu.
RemoveNetworkContainer	*NetworkDocument*	Entfernt aus dem Netzwerkdokument den übergebenen Teilgraphen.

4.1.4.6 Datenakquise

Klasse *BiochemistyQueryObject*

Die Klasse *BiochemistyQueryObject* dient zur Verwaltung von SQL-Anfragen biochemisch relevanter Anfragen. Hierzu gehören Anfragen nach Substanzen, Enzymen, Reaktionen sowie ihrer jeweiligen Synonyme. Auch Anfragen für Namen von verfügbaren Molekülstrukturen (Kapitel 3.8.6) und den dazugehörigen Molfiles sowie in einer Datenbank gespeicherten Namen von hinterlegten metabolischen Profilen (Spektren) und deren Spektren können in einem Objekt dieser Klasse gespeichert werden. Das Anzeige- und Editier-Fenster für dieses Objekt ist in Kapitel 4.1.5.15 dargestellt.

Klasse *BRENDADatabaseObject*

Die Klasse *BRENDADatabaseObject* dient als Datenbank-Schnittstelle zu biochemischen Daten. Es verwaltet die Daten über SQL-Anfragen gespeichert in dem Objekt der Klasse

BiochemistyQueryObject und liefert Datensätze zu Substanzen, Enzymen, Reaktionen, Molfiles und Spektren.

Klasse *PeriodicTable*

Die Klasse *PeriodicTable* dient als Schnittstelle für den Zugang zu atombezogenen Daten. Diese Daten werden für die Darstellung von Molekülen benutzt. *PeriodicTable* liest hierbei eine CSV-Datei ein, welche die atombezogenen Daten beinhaltet. Für einen vorgegebenen Atomnamen wird eine Farbe sowie eine Größe zurückgeben.

4.1.4.7 Import und Export

MapOmnia bietet Import- und Export-Möglichkeiten für verschiedene Graphformate. Hierbei haben drei Klassen zentrale Bedeutung: *NetworkIOHandler*, *NetworkReader* und *NetworkWriter*. Sie werden im Folgenden kurz beschrieben. Für detaillierte Informationen sei an dieser Stelle ebenfalls auf die mit dem Programm Doxygen erstellte Quelltext-Dokumentation verwiesen.

Klasse *NetworkReader*

Die Klasse *NetworkReader* ist die zentrale Klasse zum Einlesen von Graphen. Es kann aus Dateien sowie anderen I/O Devices von Qt (Basisklasse *QIODevice*) gelesen werden. Um ein Netzwerk einzulesen, wird eine *NetworkReader*-Instanz erstellt, wobei dem Konstruktor ein Dateiname oder ein Device-Pointer übergeben wird. Mit der Methode *canRead()* kann im Vorfeld überprüft werden, ob das Netzwerk eingelesen werden kann. Durch die *read()* Methode wird das Netzwerk gelesen, wobei das Netzwerk-Format automatisch erkannt wird.

Klasse *NetworkWriter*

Die Klasse *NetworkWriter* ist die zentrale Klasse, um Graphen zu schreiben. Auch hier kann sowohl in Dateien als auch in anderen I/O Devices von Qt (Basisklasse *QIODevice*) geschrieben werden. Analog zum Einlesen eines Netzwerks wird für den Schreibvorgang eine Instanz der Klasse *NetworkWriter* erstellt und dem Konstruktor ein Dateiname oder ein Device Pointer übergeben. Zusätzlich kann festgelegt werden, welches Netzwerk-Format erzeugt werden soll (*setFormat()*-Methode). Mit der Methode *write()* wird das Netzwerk schließlich geschriebenen.

Klasse *NetworkIOHandler*

Die Klasse *NetworkIOHandler* definiert eine I/O Schnittstelle für das Lesen und Schreiben von Netzwerken in MapOmnia. MapOmnia benutzt Instanzen der Klasse *NetworkReader* und *NetworkWriter*, um Netzwerke einzulesen und zu schreiben. Mit der Klasse *NetworkIOHandler* ist es möglich, eigene Klassen für verschiedene Formate zu implementieren. Um dies durchzuführen, müssen die *canRead()*- und *read()*-Methoden überschrieben und ein *NetworkIOPlugin* erstellt werden.

In Abbildung 4.6 ist das Vererbungsdiagramm der der Klasse *NetworkIOHandler* von MapOmnia gezeigt.

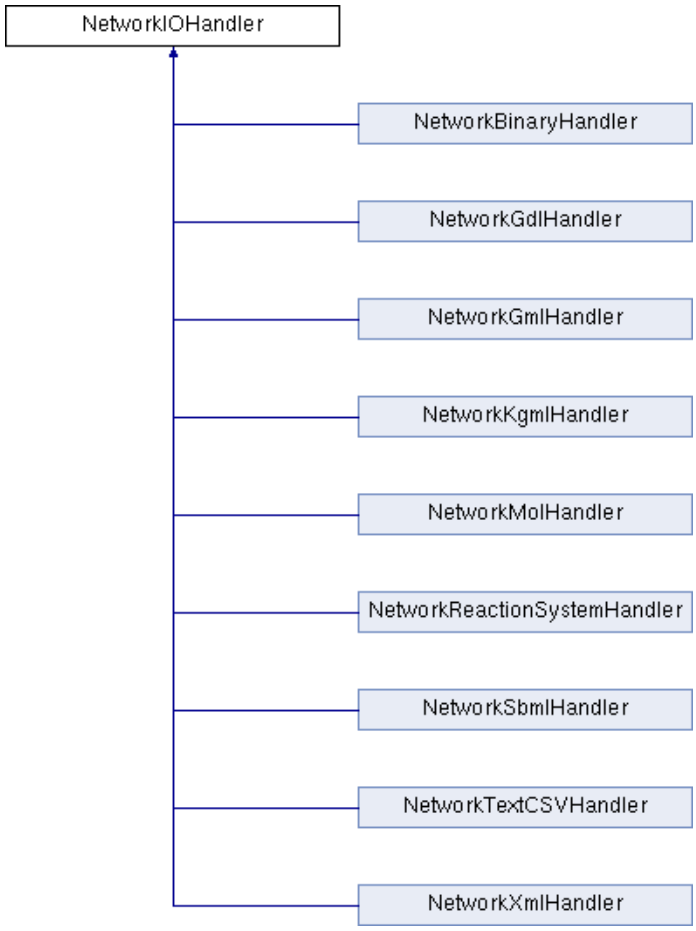

Abbildung 4.6: Vererbungsdiagramm für *NetworkIOHandler*. Für die meisten der dargestellten Handler wurde Import- sowie Export-Funktionalität implementiert. Die Namen der Handler spiegeln jeweils das unterstützte Format (GML, SBML, usw.) wieder.

4.1.5 Komponenten und Bedienung von MapOmnia

Dieser Abschnitt behandelt die grundlegenden Komponenten des Pathway-Editors MapOmnia sowie dessen Bedienung. Es wird der gesamte Funktionsumfang des Programms beschrieben sowie Kernkomponenten erläutert. In Kapitel 4.1.5.1 wird die „Network Graphics Scene“ beschrieben, welche als Container für Netzwerkobjekte fungiert. Das anschließende Kapitel 4.1.5.2 beschreibt das „Network Visualisation“-Fenster, welches die Darstellung der Szene übernimmt. Die Visualisierung des „Undo-Stack Browser“ wird in Kapitel 4.1.5.3 beschrieben. Kapitel 4.1.5.4, 4.1.5.5 und 4.1.5.6 widmen sich der Verwaltung von Netzwerken. Da MapOmnia ebenfalls Animationen der Netzwerkobjekte sowie für die darstellende Szene erstellen kann, wird diese Funktionalität in den Kapiteln 4.1.5.7 bis 4.1.5.10 erörtert. Das „Network Overview Fenster“, als Übersichtsfenster für Netzwerke, wird in Kapitel 4.1.5.11 beschrieben. MapOmnia besitzt eine eingebaute Skript-Engine und somit Skript-Funktionalität; das zugehörige Editierfenster wird in Kapitel 4.1.5.12 umrissen. MapOmnia umfasst auch Tools zur Durchführung von Netzwerkmanipulationen. Diese sind in Kapitel 4.1.5.13 aufgeführt. Das Erstellen von Abbildungen von Daten auf Netzwerkobjekte wird durch dialogbasierte Wizards unterstützt. Diese werden in Kapitel 4.1.5.14 vorgestellt. Der Aufbau der Menüleiste des Programms wird schließlich in Kapitel 4.1.5.15 aufgezeigt und abschließend werden zentrale Dialog- und Editierfenster in Kapitel 4.1.5.16 beschrieben.

4.1.5.1 Network Graphics Scene

Die „Network Graphics Scene“ dient als Container für Netzwerkobjekte, die dem Benutzer visuell dargestellt werden sollen. Für die Szene sind verschiedene Bearbeitungsmodi implementiert, welche in Tabelle 4.5 aufgelistet sind. Die Wahl des Modus kann über eine hierfür vorgesehene Symbolleiste oder in der Menüleiste erfolgen. Das Verhalten der Szene auf mausgesteuerte Nutzer-Interaktion kann über die Auswahl des Modus gesteuert werden. Sollen dem Netzwerk neue Knoten zugefügt werden, so muss die Szene auf „Node Mode“ gesetzt sein, für Kanten entsprechend auf „Edge Mode“.

Tabelle 4.5: Übersicht der Bearbeitungsmodi der Diagramszene

Bearbeitungsmodus	**Beschreibung**
Node Mode	Einfügen neuer Knoten in das Netzwerk
Edge Mode	Einfügen neuer Kanten in das Netzwerk
Select Items Mode	Selektion von Netzwerkobjekten
Manipulate Items Mode	Undo/Redo basierte Manipulation von Netzwerkobjekten
Group Mode	Keine Funktion
Structure Mode	Keine Funktion

Selektionsmöglichkeiten

Um Netzwerkobjekte zu selektieren, muss der „Select Items Mode“ gewählt werden. Die Selektion ist innerhalb der mausbasierten Selektionsauswahl auf Knoten und Kanten beschränkt. Man kann durch einen Einfachklick sowie anschließende Mausbewegung ein Auswahlrechteck zur Selektion generieren (siehe Abbildung 4.7). Möchte man die Selektion auf andere Netzwerkbereiche erweitern, so kann dieses durch Gedrückthalten der Strg-Taste erfolgen (siehe Abbildung 4.8).

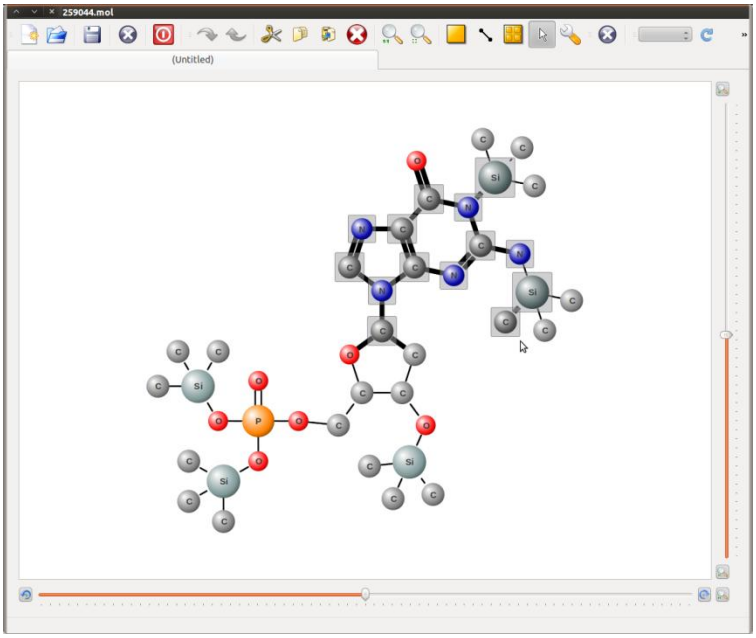

Abbildung 4.7: Im Selektionsmodus kann durch Drücken der linken Maustaste und Ziehen über einen Netzwerkbereich derselbe selektiert werden. Die selektierten Knoten werden mit einem grauen Kästchen unterlegt, die Kanten durch eine größere Strichstärke hervorgehoben.

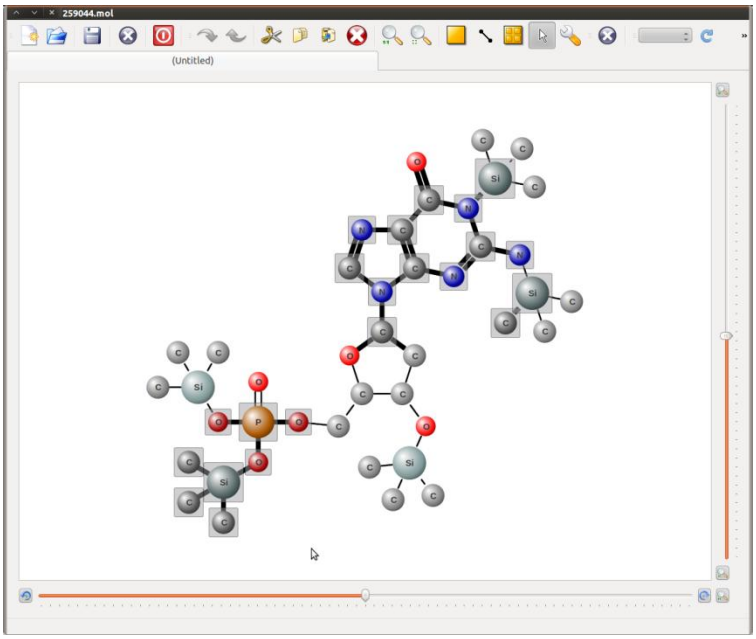

Abbildung 4.8: Eine Selektion ist durch Gedrückthalten der Strg-Taste erweitertbar.

Bearbeitungsmöglichkeiten

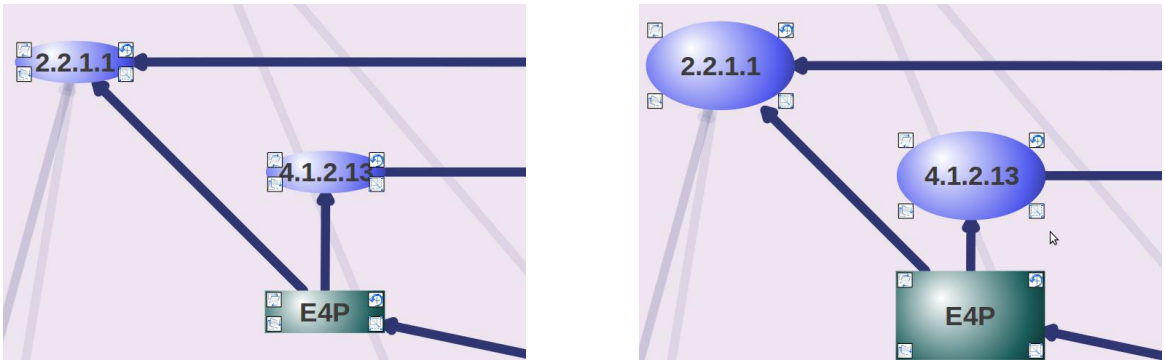

Abbildung 4.9: Befindet sich die „Network Graphics Scene“ im „Manipulate Items Mode“, so werden vier kleine Rechtecke (Handler) gezeichnet. Durch Klicken auf dieses kann z. B. die Darstellungsgröße angepasst werden (rechts). Die Größenveränderung wird automatisch für sämtliche markierten Knoten übernommen.

Veränderungen an Netzwerkobjekten können durchgeführt werden, sofern sich die Szene im „Manipulate Items Mode“ befindet. Die unter diesem Modus vorgenommenen Veränderungen werden gespeichert und dem *UndoStack* zugewiesen (siehe Kapitel 4.1.4.5). Wählt der Benutzer diesen Modus, so ändert sich automatisch das Darstellungsverhalten der Netzwerkobjekte. Es werden den Objekten kleine Rechtecke zugefügt, welche die jeweilige Veränderungsmöglichkeit visualisieren. Tabelle 4.6 gibt eine Übersicht über die implementierte Handler für die Bearbeitung von Netzwerkobjekten. Ein so modifiziertes Darstellungsverhalten ist exemplarisch in Abbildung 4.9 gezeigt. Klickt der Benutzer auf ein Rechteck (Handler), so wird dieses von dem Programm erkannt und es springt automatisch in den jeweiligen Modus für die gewünschte Bearbeitung. Netzwerkknoten können skaliert und verschoben werden. Dem „Group“- und „Structure Mode“ wurde bisher noch keine speziellen Funktionen zugeordnet.

Das Darstellungsverhalten der „Network Graphics Scene“ lässt sich ebenfalls anpassen. In dem Konfigurationsdialog (Kapitel 4.1.5.15) kann das Erscheinungsbild verändert werden. Es ist möglich, ein Hintergrundbild zu laden oder eine Brush (deutsch: der Pinsel) für den Hintergrund zu wählen. Ein Brush ist eine farbige Fläche, welche ebenfalls eine Struktur

aufweisen kann. Auch lässt sich ein Raster definieren, das seinerseits selber als Referenz für ein Layout dienen kann, sofern dieses aktiviert ist. Verschiebt man hierbei die Knoten im „Manipulate Items Mode“, so wird automatisch berechnet, welcher Rasterpunkt dem jeweiligen Knoten am nächsten liegt und dementsprechend wird dieser auf den Rasterpunkt verschoben. Dies ist in Abbildung 4.10 gezeigt.

Tabelle 4.6: Implementierte Handler für die Bearbeitung von Netzwerkobjekten

Handler	**Funktion**
Scale	Skalierung des Netzwerkobjektes
Fläche Netzwerkobjekt	Verschiebung des Netzwerkobjektes
Shear Up/Down	Keine Funktion
Shear Left/Right	Keine Funktion
Rotate	Keine Funktion

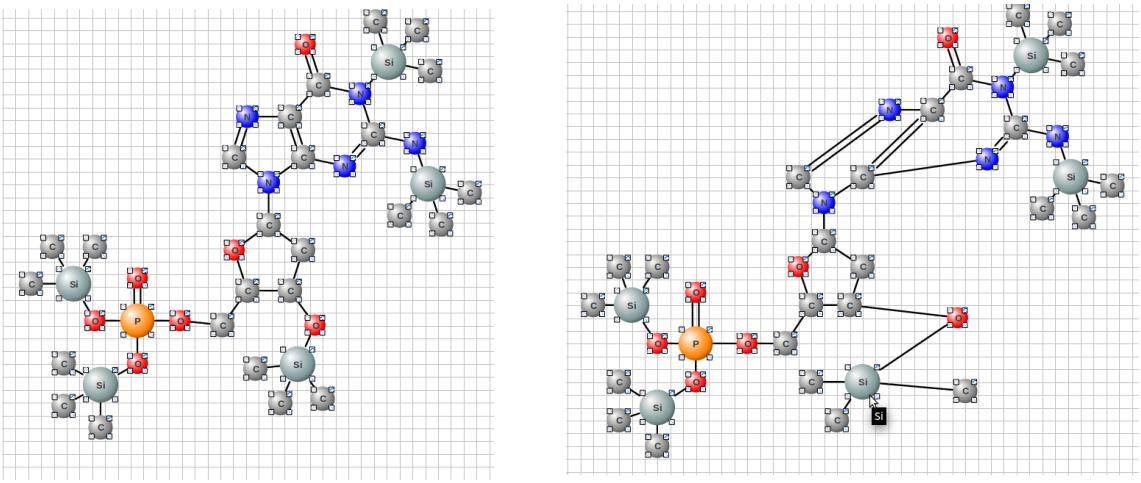

Abbildung 4.10: Befindet sich die Szene im „Manipulate Items Mode“ und ist ein Raster für das Layout definiert, so wird beim Verschieben der Knoten, diese automatisch zum jeweils nächsten Rasterpunkt verschoben (rechts).

Um die Anwendung auch bei einer großen Anzahl von Netzwerkobjekten performant zu halten, wurde ein zoomabhängiges Darstellungsverhalten implementiert. Es berücksichtigt

in drei Stufen den Detailgrad der Objekte (siehe Abbildung 4.11). In der ersten Stufe werden Netzwerkobjekte in sämtlichen Details dargestellt. Dies ist sinnvoll, wenn man sehr nah in die Szene zoomt (siehe Abbildung 4.11, a). Zoomt der Benutzer heraus, so hat dieses zur Folge das immer mehr Netzwerkobjekte von dem „Network Visualisation Fenster“ dargestellt werden müssen und zusätzlich werden die Objekte kleiner. Daher wird nicht mehr die maximale Detailtiefe benötigt. Die Netzwerkobjekte werden entsprechend gerendert (siehe Abbildung 4.11, b). Bei sehr kleiner Zoomstufe werden die Objekte in der dritten Detailstufe gerendert dargestellt. Hier brauchen kaum noch Details visualisiert werden (siehe Abbildung 4.11, c).

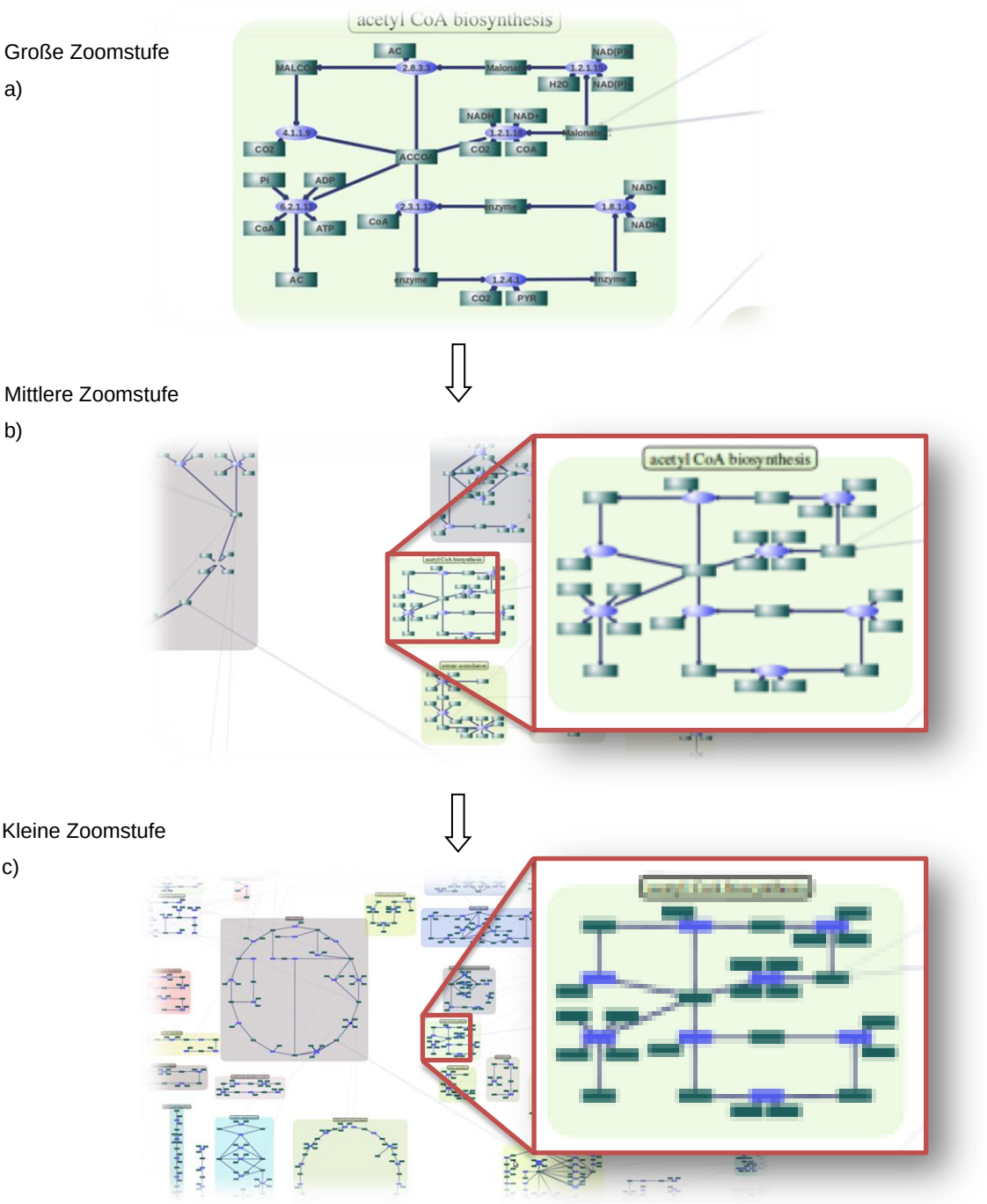

Abbildung 4.11: Der Detailgrad der Netzwerkobjekte Knoten, Kanten und Gruppen hängt von der Zoomstufe ab. In a) (große Zoomstufe) ist maximaler Detailgrad vorhanden. In b) (mittlere Zoomstufe) werden schon keine Beschriftungen mehr dargestellt. In c) (kleine Zoomstufe) werden aufwendige Grafikeffekte, wie Gradientendarstellung, ausgeschaltet.

4.1.5.2 Network Visualisation Fenster (Klasse *GraphicsViewTransformWidget*)

Im mittleren Fenster der MapOmnia-GUI (Abbildung 4.2, a), dem Multiple-Document-Interface, werden die aktuell ausgewählten Netzwerke dargestellt. Hierbei wird für jeden neuen Graphen eine Instanz der Klasse *NetworkDocument* (siehe Kapitel 4.1.4.2) mit einem neuen „Network Visualisation Fenster“ (Klassenname: *GraphicsViewTransformWidget*) erstellt, in welchem die „Network Graphics Scene“ mit den in ihr befindlichen Netzwerkobjekten visualisiert wird. Abbildung 4.12 zeigt ein solches Fenster exemplarisch.

Navigationsmöglichkeiten

Das Fenster selber bietet Navigationsmöglichkeiten für die Szene über kleine Tool-Buttons (deutsch: die Anwendungsschaltflächen) sowie Slider- (deutsch: der Schieber) und Navigationsbalken an. Der Benutzer kann in diesem Fenster in die Szene hinein- und aus dieser herauszoomen, die Szene rotieren und die Markierung von Netzwerkobjekten mithilfe der Maus innerhalb der „Network Graphics Scene“ durchführen.

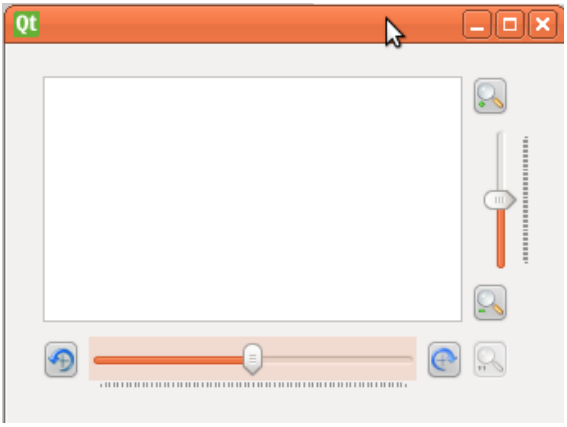

Abbildung 4.12: Das „Network Visualisation Fenster“ realisiert die Visualisierung der „Network Graphics Scene“ sowie der Interaktionsmöglichkeiten mit dieser. Am Rand befinden sich grafische Tool-Buttons, welche das Rotations- sowie Zoomverhalten steuern. Über Navigationsbalken kann innerhalb der Szene in x- und y-Richtung navigiert werden.

4.1.5.3 Undo-Stack Browser

Der „Undo-Stack Browser“ fungiert als grafisches Interface für die Undo/Redo-Funktionalität des Programmes. Der in 4.1.4.5 beschriebene Stack wird hier als Historie von Einzelaktivitäten dargestellt (Abbildung 4.13). Innerhalb dieser Liste kann mit der Maus navigiert werden und ein vorheriger Zustand des Netzwerkes kann wiederhergestellt werden.

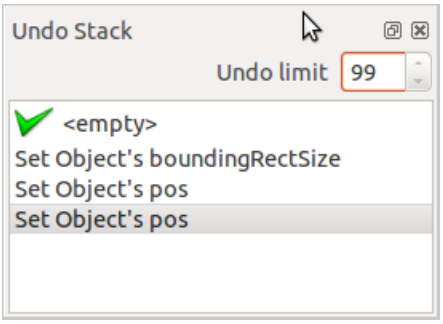

Abbildung 4.13: Der „Undo-Stack Browser“ ermöglicht es, Veränderungen am Netzwerk rückgängig zu machen bzw. wiederherzustellen. Diese Veränderungen werden als Historie dargestellt und aussagekräftig dokumentiert.

4.1.5.4 Network Data Browser

Das Docking-Fenster „Network Data Browser“ (siehe Abbildung 4.14) ist zweigeteilt. Im linken Teil befindet sich der „Network Data Tree Browser" (siehe Kapitel 4.1.5.5), in dem die gespeicherten Daten der geöffneten Netzwerke als Baum dargestellt werden. Der rechte Teil wird dynamisch erstellt, abhängig von der Selektion im linken Fenster. In MapOmnia sind für diesen Bereich drei Darstellungsfenster implementiert: „Network Data Table Browser“ (siehe Kapitel 4.1.5.6), „Network Object Animation Browser“ (siehe Kapitel 4.1.5.8) und „Network Scene Animation Browser“ (siehe Kapitel 4.1.5.10).

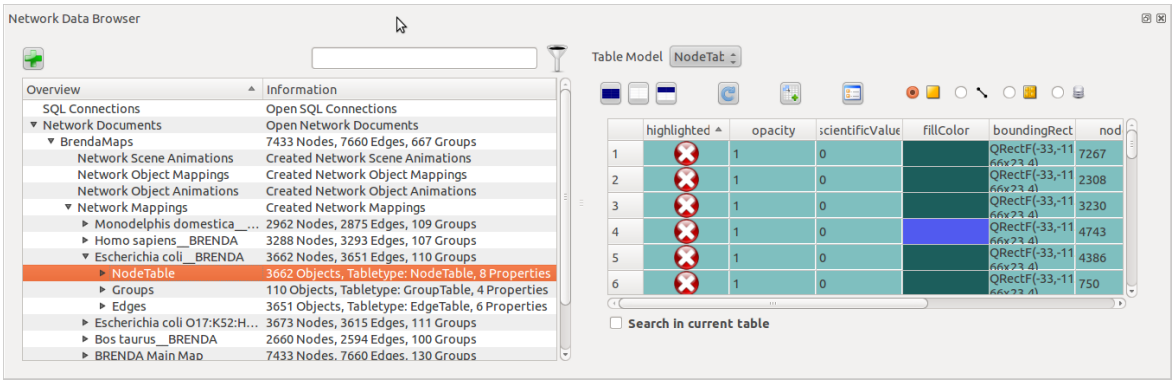

Abbildung 4.14: Der „Network Data Browser“ gibt dem Benutzer die Möglichkeit, zwischen Netzwerken, Teilnetzwerken und Animationen zu wechseln.

4.1.5.5 Network Data Tree Browser

Der „Network Data Tree Browser" stellt die geöffneten Verbindungen zu SQL-Datenbanken dar sowie die in einem Netzwerk-Dokument gespeicherten Objekte. Hierzu gehören sowohl die im Netzwerk befindlichen Knoten, Kanten und Gruppen, die verschiedenen Teilnetzwerke als auch erstellte Animationen für Netzwerkobjekte und die Netzwerkszene.

In der obersten Ebene des Baums werden die geöffneten SQL-Datenbanken sowie Netzwerke angezeigt. Jedes geöffnete Netzwerk hat seinerseits wieder 5 Kategorien als Kindknoten. Wurden „Network Object Animation“-Objekte erstellt (siehe 4.1.5.7), sind diese unter dem Baumknoten „Network Object Animations“ gelistet. Der Knoten „Network Object Mappings“ beinhaltet geöffnete Dateien, welche dem *NetworkPropertyStorage* (siehe 4.1.4.2) zugrunde liegen. Unter „Network Mappings“ werden alle Teilnetzwerke des Gesamtnetzwerkes als Kindknoten abgelegt. Diese Teilnetzwerke beinhalten ihrerseits Knoten, Kanten sowie Gruppen. Hierbei wird der Baum so erweitert, dass für diese Netzwerkobjekte ebenfalls Kindknoten erstellt werden.

Der Knoten „Main Network“ selbst repräsentiert das Gesamtnetzwerk. Dieses besteht aus sämtlichen Knoten, Kanten und Gruppen der Einzelnetzwerke. Damit diese sichtbar bleiben, wird in diesem Netzwerk automatisiert die Sichtbarkeit der Netzwerkobjekte aktiviert sowie deren Transparenz deaktiviert. Der „Default Network“-Knoten repräsentiert

ebenfalls das Gesamtnetzwerk, allerdings mit dem Unterschied, dass hier die Transparenz der Netzwerkobjekte 80 % beträgt. Dieses Netzwerk fungiert als Referenznetzwerk.

Alle Instanzen der Klassen, die in diesem Baum für die Auswahl dargestellt werden, besitzen eine *getInformation()*-Methode. Über diese Methode können Informationen für das jeweilige erzeugte Objekt abgerufen werden. So gibt z. B. ein *NetworkContainer*-Objekt (siehe Kapitel 4.1.4.2) die Information zurück, wie viele Knoten, Kanten und Gruppen in ihm gespeichert sind. Alle diese Informationen werden in der zweiten Spalte „Information“ des Views angezeigt. Abbildung 4.15 zeigt das Erscheinungsbild des „Network Data Tree Browser“.

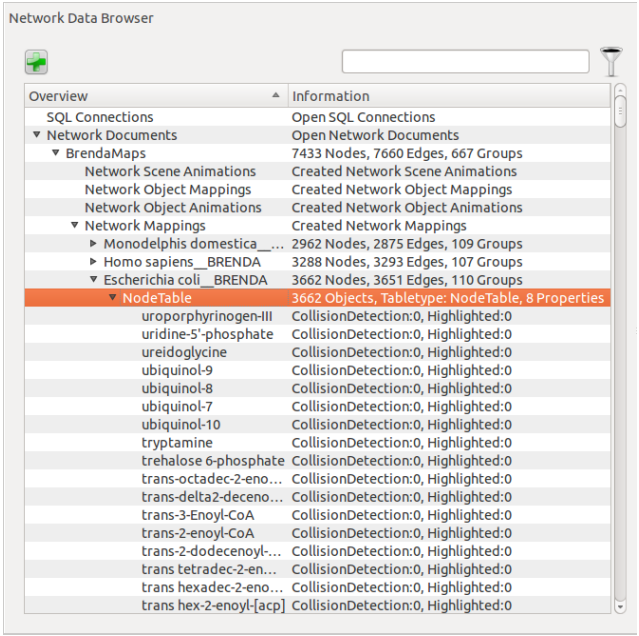

Abbildung 4.15: Der „Network Data Tree Browser“ besteht aus zwei Spalten. In der ersten wird der Baum aufgebaut, welcher die vorhandenen Datenbankanbindungen, geöffnete Netzwerke und Animationen hierarchisch darstellt. Die zweite Spalte zeigt dem Benutzer zusätzliche Informationen.

Navigationsmöglichkeiten

Die Bedienung dieser Ansicht des Baums kann über die Tastatur-Pfeiltasten und über die Maus erfolgen. Es wird erkannt, welches Objekt dem selektierten Baumknoten zugeordnet ist. Hierdurch ist es möglich, für selektierte Netzwerkobjekte eine Navigation innerhalb der Netzwerke anzubieten. Wird ein Knoten oder eine Kante ausgewählt, fokussiert die Szene

automatisch auf das gewählte Objekt. Wird eine Gruppe ausgewählt, passt sich die Ansicht zusätzlich der Größe an. In Abbildung 4.16 wird das Fokussierungsverhalten exemplarisch durch Auswahl eines Knotens gezeigt.

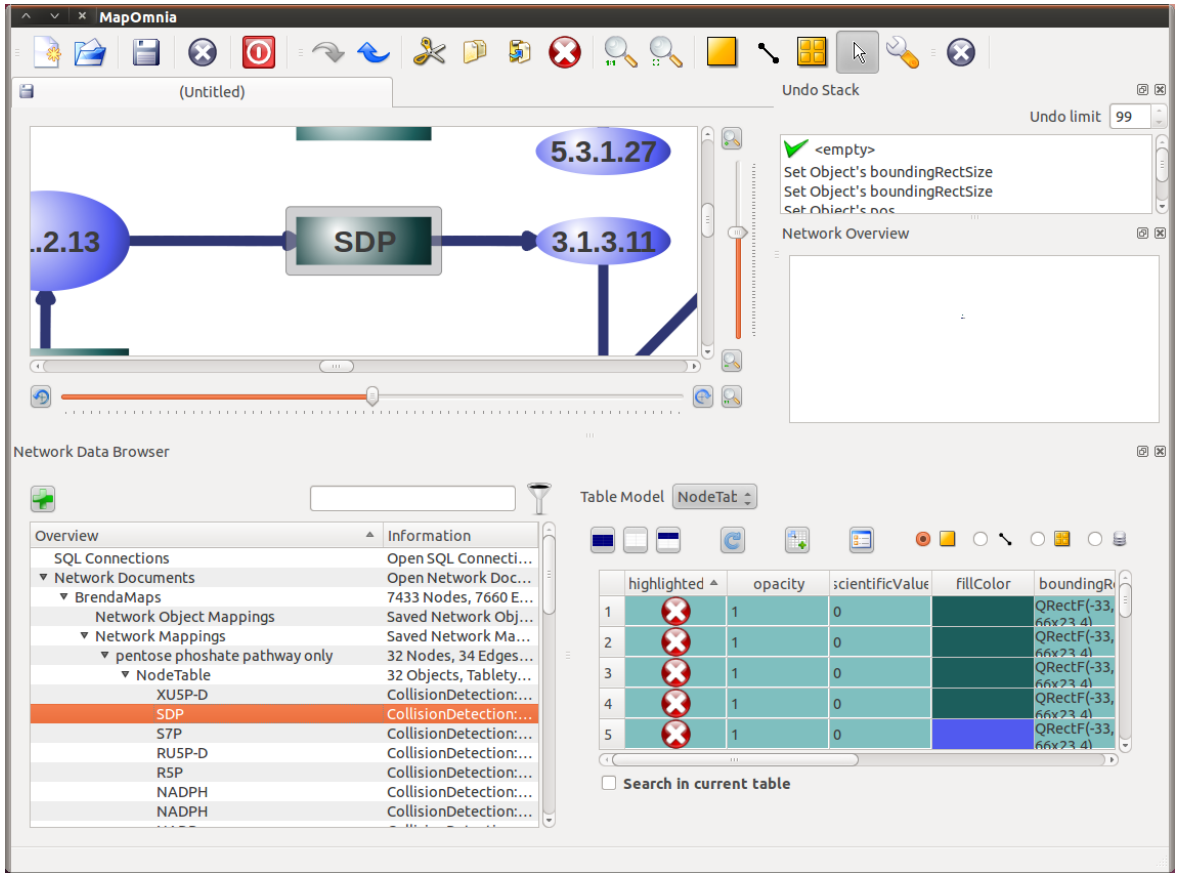

Abbildung 4.16: Bewegt man sich innerhalb des Baumes, so wird bei der Auswahl von Knoten, Kanten und Gruppen automatisch auf diese fokussiert. Im oberen Beispiel wurde SDP (D-sedoheptulose 1,7-bisphosphate) ausgewählt.

Wird erkannt, dass es sich bei der Selektion um ein Teilnetzwerk handelt, wird das „Network Visualisation Fenster“ aktualisiert, indem das Teilnetzwerk angezeigt wird. Abbildung 4.17 zeigt ein solches Verhalten exemplarisch.

Es kann hierbei zwischen den erstellten Teilnetzwerken gewechselt werden. Die Teilnetzwerke können selektiert werden, und die Netzwerkobjekte werden dann direkt mit den Daten überschrieben, die in dem zugehörigen *NetworkContainer*-Objekt gespeichert

sind (siehe Kapitel 4.1.4.2). Hierbei sind selbst bei großen Netzwerken (10000 Knoten) kaum Latenzen (ca. 100 ms) zu verzeichnen.

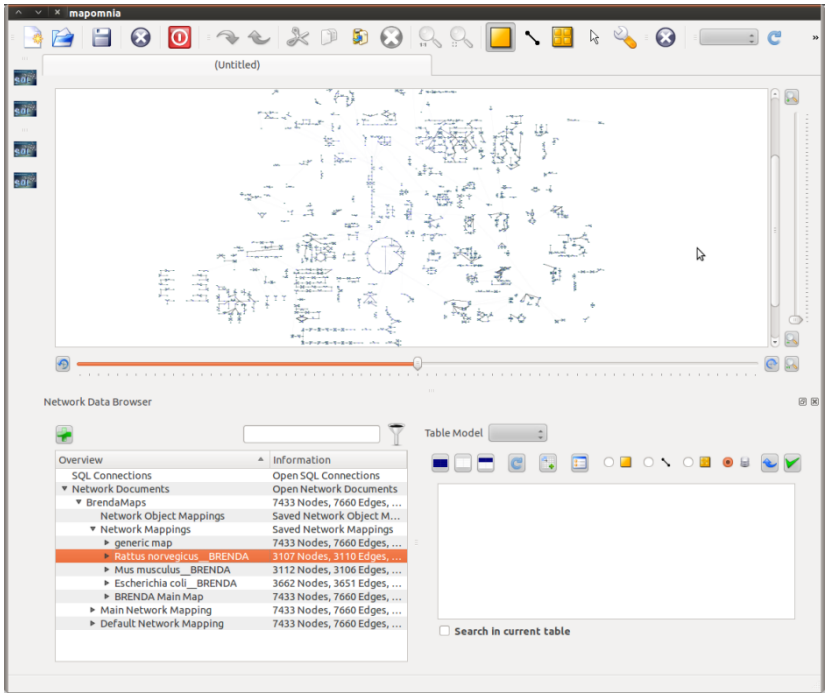

Abbildung 4.17: Ansicht eines Teilnetzwerkes: Durch Navigation in der Netzwerkhierarchie können Teilnetzwerke des Gesamtnetzwerks ausgewählt werden. In der obigen Grafik wurde das Netzwerk für *Rattus norvegicus* gewählt, welches ein Teilnetzwerk von BRENDA-Maps darstellt.

Neben den erstellten Teilnetzwerken werden die erstellten Animationen der Netzwerkobjekte sowie der Netzwerkszene in dem Auswahlbaum angezeigt. Wählt der Benutzer einen Baumknoten, der eine Animation (Kapitel 4.1.5.7, 4.1.5.9) repräsentiert, so wird im rechten Teil des „Network Data Browsers“ das jeweilige Fenster sichtbar, welches für den Animationstyp vorgesehen ist und wodurch sich die gewählte Animation steuern lässt. Die Erstellung von Animationen wird unter 4.1.5.8 und 4.1.5.10 näher erläutert.

Selektionsmöglichkeiten

Der Baum unterstützt Einzelauswahl, d.h., es können nur einzelne Netzwerkobjekte (Knoten, Kanten und Gruppen) und Teilnetzwerke gewählt werden.

Sortier- und Filtermöglichkeiten

Die Ansicht des Baums selber bietet die Möglichkeit der spaltenweisen Sortierung. Es kann also eine alphabetische Sortierung der jeweiligen Baumebenen erfolgen, was zur Folge hat, dass die Netzwerkobjekte selber alphabetisch sortiert angezeigt werden.

4.1.5.6 Network Data Table Browser

Der „Network Data Table Browser“ dient der Darstellung der *NetworkContainer* (siehe 4.1.4.2) sowie der *NetworkPropertyStorage* (ebenfalls in Kapitel 4.1.4.2). Wird im „Network Data Tree Browser“ ein Baumknoten ausgewählt, der für ein *NetworkPropertyStorage* steht, so wird der „Network Data Table Browser“ aktualisiert. In diesem Fall wird die Datenmatrix des Datenmodells (vergleiche Kapitel 3.7.2.3 und 4.1.4.2) in diesem Fenster dargestellt, welche die in dem Modell gespeicherten Dateninformationen enthält. Außerdem werden in dem „Network Data Table Browser“ die Attribute der Netzwerkobjekte als Datenmatrix angezeigt, die im „Network Visualisation Fenster“ selektiert wurden. Wie im Kapitel 3.7.2.3 beschrieben, erlaubt das verwendete Datenmodell die Speicherung diverser Datenformate. Es lassen sich Textformate, Zahlenformate, Bildformate und vieles mehr in dieser Datenmatrix speichern, die dann in dem „Network Data Table Browser“ angezeigt werden (siehe Abbildung 4.18). Für das gewünschte Darstellungsverhalten sorgt das in Kapitel 4.1.4.3 beschriebene Delegate.

Selektionsmöglichkeiten

Der Benutzer kann Reihen in der Datenmatrix durch die Tastatur-Pfeiltasten und die Maus auswählen. Hierbei bewirkt das Drücken der Strg-Taste eine durchgehende Selektion. Das Drücken der Shift-Taste bewirkt eine unterbrochene Selektion. Darüber hinaus bietet das Fenster kleine Tool-Buttons an: Mit diesen kann die gesamte Tabelle selektiert, deselektiert und die aktuelle Selektion umgeschaltet werden. Die geladene Tabelle steht hierbei in Kommunikation mit dem „Network Visualisation Fenster“. So werden auch hier die jeweiligen Netzwerkobjekte selektiert dargestellt, welche in der Datenmatrix ausgewählt sind.

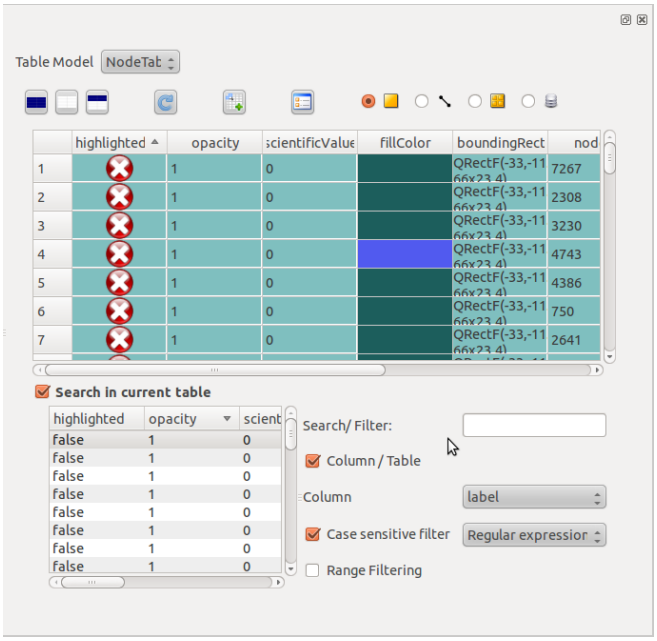

Abbildung 4.18: Der „Network Data Table Browser“ dient der Darstellung der Datenmatrix, die für die Netzwerkobjekte innerhalb des *NetworkPropertyStorage* (Kapitel 4.1.4.2) gespeichert sind. Durch das verwendete *PropertyItemDelegate* (Kapitel 4.1.4.3) werden die Zellen der Datenmatrix wie gewünscht dargestellt.

Bearbeitungsmöglichkeiten

Bei Doppelklick in ein Datenfeld wird automatisch der für den Datentyp vorgesehene Editor aufgerufen, z. B. für Textdaten ein Text-Browser, in dem der Text bearbeitet werden kann. Eine abgeschlossene Manipulation wird direkt in dem dazugehörigen Netzwerkobjekt gespeichert. Diese Editierung der Netzwerkobjekte erfolgt jedoch nicht Undo-basiert, d.h., sie kann nicht per Undo-Befehl rückgängig gemacht werden.

Außerdem kann die gespeicherte Datenmatrix erweitert oder verkleinert werden. Es können dialogbasiert neue Attribute dazu- bzw. abgewählt werden. Hierbei werden auch sämtliche dynamischen Attribute zur Selektion angeboten.

Navigationsmöglichkeiten

Neben der Selektion wird ebenfalls zu den selektierten Netzwerkobjekten navigiert, sofern dieses Verhalten gewünscht und aktiviert ist. Wird hierbei mehr als eine Reihe innerhalb

der Datenmatrix ausgewählt, so wird auch in dem „Network Visualisation Fenster“ dementsprechend auf die Gesamtheit der selektierten Netzwerkobjekte fokussiert.

Sortier- und Filtermöglichkeiten

Ein wichtiges Feature ist die Möglichkeit, in der Datenmatrix nach Vorkommen von Daten zu filtern. So wurde ein Proxy-Modell implementiert, das Sortier- und Filtermöglichkeiten bietet (siehe Kapitel 4.1.4.4). Der Benutzer kann die gesamte Tabelle durchsuchen oder sich für eine bestimmte Spalte entscheiden. Es kann hierbei zwischen der Suche nach regulärem Ausdruck sowie String-basierter oder Wildcard-basierter Suche gewählt werden. Das Filterergebnis wird in einem weiteren Fenster angezeigt (siehe Abbildung 4.18 unten links). Hier kann der Benutzer die Reihen selektieren, wodurch die Auswahl der gefilterten und selektieren Netzwerkobjekt erfolgt.

4.1.5.7 Network Object Animation

Um Veränderungen in Teilnetzwerken auf besondere Weise sichtbar zu machen, gibt es die Möglichkeit, hierfür Animationen zu erstellen. Wurde z. B. eine metabolische Stoffwechselkarte geladen und ein Teilnetzwerk erstellt, in dem sämtlich ATPs rot markiert sind, so kann durch eine Animation mit z. B. unbegrenzter Wiederholung (englisch: Loop) ein Blinkeffekt dargestellt werden. Auch ist es möglich Flussveränderung, d. h. die Dicke der Kanten variiert, von solchen Netzwerken dynamisch zu visualisieren.
Die Erstellung von Animationen ist nicht auf bestimmte Attribute der Netzwerkobjekte beschränkt. Durch Verwendung des Qt Property Systems (siehe Kapitel 3.7.2.1) in Verbindung mit der Speicherung durch das *QVariant* basierte Datenmodell (siehe Kapitel 3.7.2.3) kann zwischen sämtlichen Attributen interpoliert werden, für die Interpolatoren registriert sind oder von Qt bereitgestellt werden.
Die Erstellung einer „Network Object Animation“ erfolgt dialogbasiert. Innerhalb des Dialoges können die Parameter für die Animation ausgewählt werden.
Hierzu gehören: die Animationslänge (englisch: Duration), die Anzahl der Wiederholungen (englisch: Loop), die Interpolationsfunktion (englisch: Easing Function), das Start- und Endteilnetzwerk sowie die Auswahl der Objekte über die interpoliert werden soll. Die Auswahl der Wiederholungsrate (englisch: Frames per Second; FPS) sowie der Geschwindigkeit (englisch: Speed) hat noch keine Funktion und ist für eventuellen Export

als Videosequenz vorgesehen. Auf den Tabs „Property Nodes“, „Property Edges“ und „Property Groups“ können die zu verwendenden Attribute für die jeweiligen Netzwerkobjekte ausgewählt werden. Abbildung 4.19 zeigt das Dialogfenster für die Erstellung einer Animation.

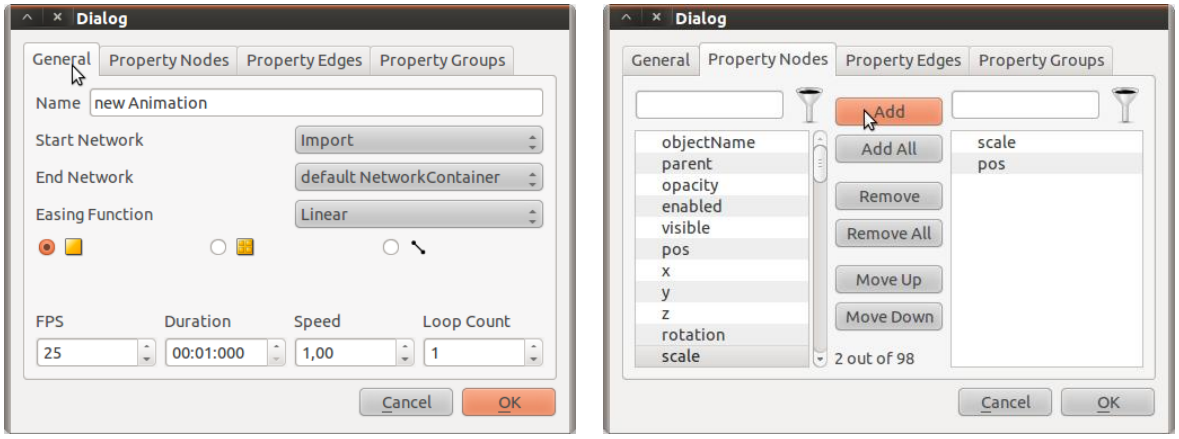

Abbildung 4.19: Das Erstellen einer Animation erfolgt dialogbasiert. Das Dialogfenster ist in vier Tabs gegliedert. Über das erste werden Start sowie End-Teilnetzwerk festgelegt und die Interpolationsfunktion sowie Animationslänge gewählt. Über die weiteren Tabs können Attribute der Netzwerkobjekte ausgesucht werden, welche für die Animation berücksichtigt werden sollen.

4.1.5.8 Network Object Animation Browser

Wird im „Network Data Tree Browser“ ein Kindknoten ausgewählt, der eine „Network Object Animation“ repräsentiert, so wird auf der rechten Seite des „Network Data Browser“ das Kontrollfenster für diese Animation bereitgestellt. Hier kann die Animation gestartet, gestoppt und pausiert werden. Abbildung 4.20 zeigt dieses Kontrollfenster.

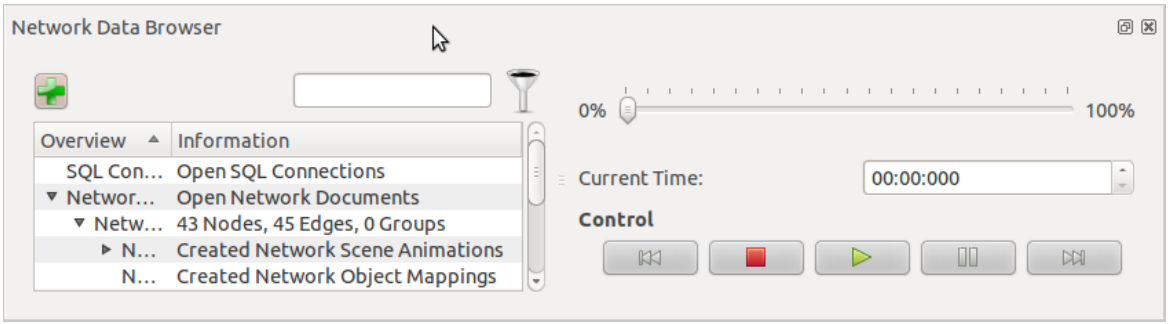

Abbildung 4.20: Der „Network Object Animation Browser“ auf der rechten Seite bietet Bedienelemente wie Start, Stopp und Pause für die Ablaufkontrolle einer erstellten Animation.

4.1.5.9 Network Scene Animation

Neben der „Network Object Animation“, bietet MapOmnia auch die Möglichkeit, eine „Network Scene Animation“ zu erstellen. Bei dieser Animation wird eine Art Route innerhalb der Szene durch die Auswahl von Netzwerkobjekten vorgegeben. Hierbei navigiert der Fokus der Szene von Netzwerkobjekt zu Netzwerkobjekt. Hat der Benutzer z. B. innerhalb eines Stoffwechselpfades über das Tool „Shortest Path“ (Kapitel 4.1.5.13), einen kürzesten Pfad berechnen lassen, so kann er hierfür eine Animation erstellen, um entlang dieses Weges zu navigieren. Analog zu der Erstellung einer „Network Object Animation“ erfolgt dies ebenfalls dialogbasiert. Innerhalb dieses Dialoges können wiederum die Parameter für die Animation eingestellt werden.

Hierzu gehören: die Animationslänge (englisch: Duration), die Anzahl der Wiederholungen (englisch: Loop), die Interpolationsfunktion (englisch: Easing Function), die Auswahl der Objekte die für die Animation berücksichtigt werden sollen, die Richtung der Animation sowie die Verweildauer auf jedem der ausgewählten Netzwerkobjekte. Die Auswahl der Wiederholungsrate (FPS) sowie der Geschwindigkeit (englisch: Speed) hat auch hier keine Funktion und ist ebenfalls für den Export als Videosequenz vorgesehen. Abbildung 4.21 zeigt das Dialogfenster für die Erstellung einer solchen Animation.

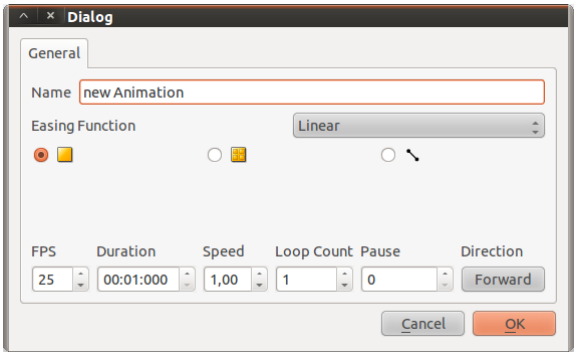

Abbildung 4.21: Das Erstellen einer „Network Scene Animation“ erfolgt dialogbasiert. Es wird die Interpolationsfunktion gewählt, die Animationslänge, die Richtung der Animation und die Wiederholungsanzahl. Die Pausenlänge gibt vor, wie lange auf dem jeweiligen Netzwerkobjekt verweilt werden soll. Außerdem kann bestimmt werden, welche Typen an Netzwerkobjekten berücksichtigt werden sollen.

4.1.5.10 Network Scene Animation Browser

Bei Auswahl einer „Network Scene Animation“ wird auf der rechten Seite (vergleiche Abbildung 4.22) der „Network Scene Animation Browser“ dargestellt. Über dieses Bedienfenster hat der Benutzer Zugriff auf sämtliche Konfigurationsmöglichkeiten der erstellten Animation und kann diese steuern. Sämtliche Einstellmöglichkeiten, wie in Kapitel 4.1.5.9 beschrieben, können auch in diesem Bedienfenster modifiziert werden.

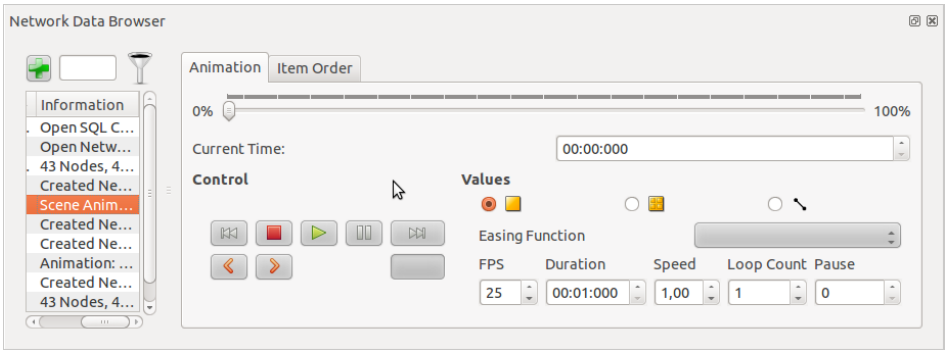

Abbildung 4.22: Der „Network Scene Animation Browser“ auf der rechten Seite bietet ebenso wie der „Network Object Animation Browser“ Bedienelemente wie Start, Stopp und Pause zur Ablaufkontrolle einer erstellten Animation an. Darüber hinaus werden Veränderungen aller typischen Animationsparameter angeboten. Im zweiten Tab „Item Order“ kann die Reihenfolge der Netzwerkobjekte angepasst werden.

4.1.5.11 Network Overview Fenster

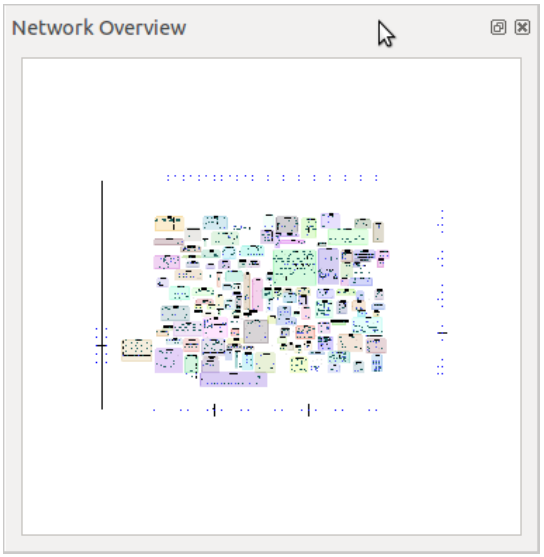

Abbildung 4.23: Das „Network Overview Fenster“ gibt einen Gesamtüberblick über das ausgewählte Netzwerk. Durch Mausklick in dieses Fenster wird der Fokus im „Network Visualisation Fenster“ festgelegt.

Das „Network Overview Fenster“ zeigt eine miniaturisierte Übersichtskarte (Abbildung 4.23), die dem Benutzer als weitere Navigationshilfe dienen soll. Es wird hierbei eine verkleinerte Version des geöffneten Netzwerkes als Gesamtansicht dargestellt.

Navigationsmöglichkeiten

Die Übersichtskarte ist mit der Netzwerkansicht (Kapitel 4.1.5.2) gekoppelt. Durch Klicken in die miniaturisierte Karte wird dieser Punkt auf die große Karte transformiert, wodurch sich die Position im „Network Visualisation Fenster“ ändert.

4.1.5.12 Network Script Fenster

Die im Kapitel 3.7.2.4 beschriebene Skript-Funktionalität von Qt wurde für MapOmnia realisiert. Sie ermöglicht es automatisierte Tasks, Netzwerksuchen oder sonstige Algorithmik für ein Netzwerk durchführen zu können. Es ist ebenfalls möglich, ein Netzwerk aus einem Skript generieren zu lassen oder bestimmte Vor-Einstellungen auf das Erscheinungsbild des Netzwerkes anzuwenden.

Der Nutzer kann die Skript-Funktion über ein eigenes Fenster „Network Script“ nutzen. In diesem können sowohl Skript-Funktionen neu erstellt als auch vorhandene aus dem Dateisystem oder über das Hauptmenü „Recent Script“ geladen werden. Der Texteditor unterstützt Undo/Redo-Funktionalität. Ebenfalls können erstellte Skripte unter Angabe eines Skript-Namens im Dateisystem bzw. in dem „Recent Script“-Menü gespeichert werden.
Über die Skript-Engine verfügbar und dadurch innerhalb eines Skripts ansprechbar, sind Instanzen des *GraphicsViewTransformWidget* (Kapitel 4.1.5.2), der *Klasse NetworkDocument* (Kapitel 4.1.4.2) sowie sämtliche Netzwerkobjekte (Kapitel 4.1.4.1), sofern sie über den Menüpunkt „Add All To Script Engine“ bzw. „Add Selection“ innerhalb der Skript-Engine registriert wurden.
Im Anhang sind in den Tabellen A.2 bis A.7 sämtliche Methoden in Abhängigkeit des Objekts aufgezeigt, welche durch das Skriptfenster ansprechbar sind. Das „Network Script Fenster“ bietet darüber hinaus auch eine Autovervollständigungs-Funktion an für:

- Die Skript-Namen registrierter Objekte
- Die Methoden des *NetworkGraphicsWidget* (siehe Tabelle A.2 im Anhang)
- Die Methoden des *NodeObject* (siehe Tabelle A.3 im Anhang)
- Die Methoden des *EdgeObject* (siehe Tabelle A.4 im Anhang)
- Die Methoden des *GroupObject* (siehe Tabelle A.5 im Anhang)
- Die Methoden des *GraphicsViewTransformWidget* (siehe Tabelle A.6 im Anhang)
- Die Methoden des *NetworkDocument* (siehe Tabelle A.7 im Anhang)

Um die Übersichtlichkeit der erstellten Skripte zu erhöhen, wurde für die oben aufgeführten Objekte und Methoden eine Hervorhebung der Syntax (englisch: Syntax-Highlighting) mit unterschiedlicher Farbgebung implementiert.

Abbildung 4.24 zeigt eine Ansicht des Skript-Fensters, in das ein Anwendungsbeispiel geladen wurde.

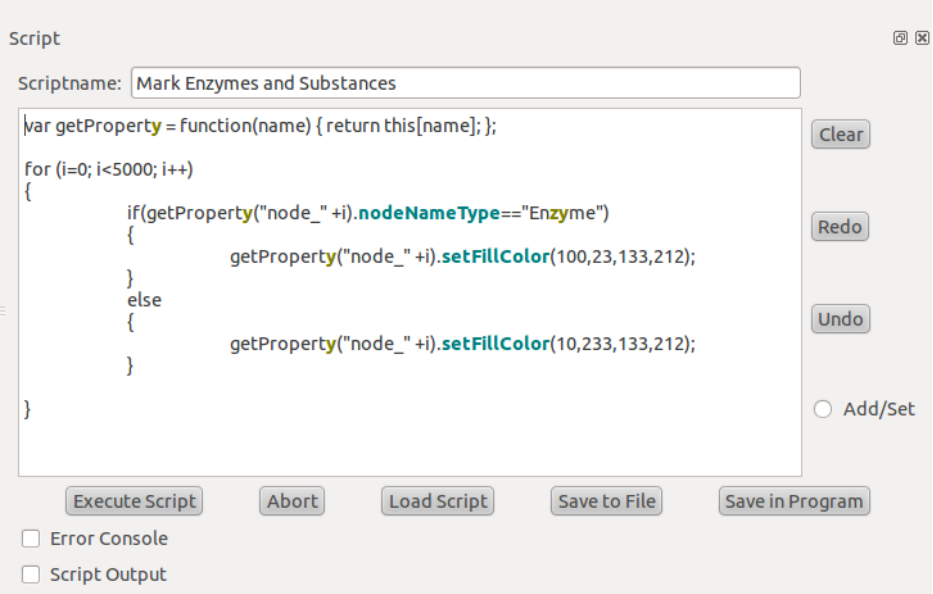

Abbildung 4.24: Das „Network Script Fenster“ bietet die Möglichkeit, Skripte zu erstellen oder vorhandene zu laden. Werden diese ausgeführt, wird das Netzwerk entsprechend manipuliert.

Selektions-, Bearbeitungs-, Navigations- und Sortier- und Filtermöglichkeiten

Da sehr viele Funktionen der Netzwerkobjekte sowie der Instanz der Klasse *GraphicsViewTransformWidget* und der Klasse *NetworkDocument* aus dem Skript heraus ansprechbar sind, werden sämtliche Selektions-, Bearbeitungs-, Navigations- und Sortier- und Filtermöglichkeiten offeriert, die auch im Programm MapOmnia implementiert sind. Darüber hinaus erkennt die Skript-Engine den Skript-Namen eines Netzwerkobjektes und springt automatisch innerhalb des „Network Visualisation Fenster“ auf dieses Netzwerkobjekt, sofern innerhalb des Texteditors auf diesen Namen geklickt wird.

4.1.5.13 Tools

MapOmnia bietet einige Tools. Hierzu gehört neben verschiedenen Analysemethoden auch unter anderem das TCP/IP Server Projekt, das ausführlich in Kapitel 4.2 beschrieben wird. Im Folgenden werden die Tools kurz dargestellt.

Tool „Decomposition“

Bestätigt man „Decomposition“ im Menü „Tools“, so wird ein neues Teilnetzwerk Undo-basiert erstellt. Es werden für das zu dem Zeitpunkt selektierte Netzwerk die jeweiligen Knotengrade (Kapitel 2.1.2.1) ermittelt und aufgeteilt. Dadurch entstehen im neu erstellten Teilnetzwerk 4 Gruppen: „Linear Nodes“ (Knoten mit Knotengrad gleich 2), „Dead End Nodes“ (Knoten mit Knotengrad gleich 1), „Hub Nodes“ (Knoten mit Knotengrad größer 2) und „Not connected Nodes“ (Knoten mit Knotengrad gleich 0). Die Knoten werden nachfolgend entsprechend ihrer Knotengrade zu den jeweiligen Gruppen hinzugefügt. Das Dekompositionsergebnis ist in Abbildung 4.25 ist exemplarisch für das Molekül 2'-Deoxyguanosine_5'-p_5TMS gezeigt.

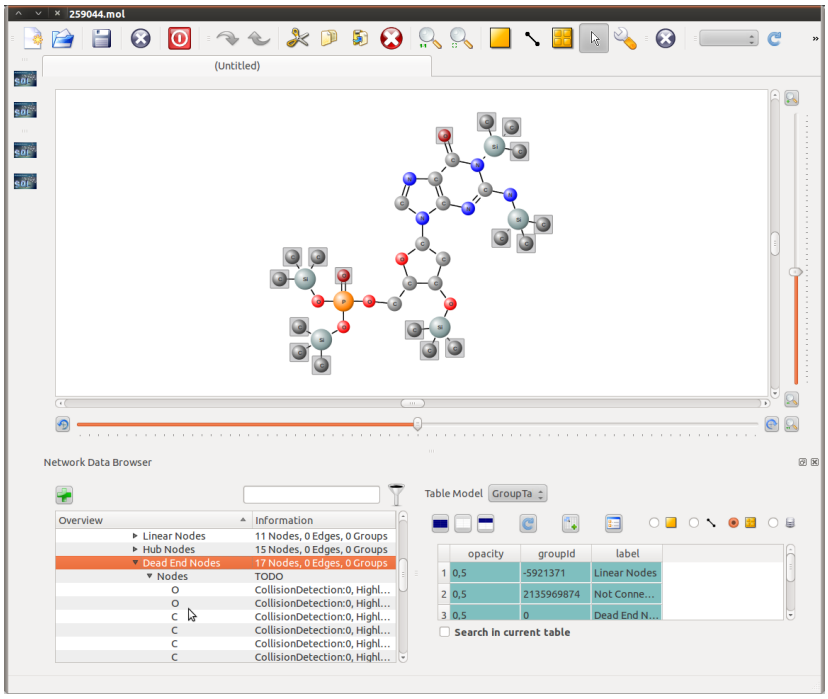

Abbildung 4.25: Nach Ausführung der Dekomposition werden die Knoten nach ihren Knotengrad separiert. Im gezeigten Beispiel wurde die Gruppe der „Dead End Nodes“ ausgewählt, welche grau hinterlegt hervorgehoben werden.

Tool „HTTP Daemon“

Über das Menü „Tools“ kann für ein geöffnetes Netzwerk ein TCP/IP Dämon erstellt werden. Der Dämon läuft im Server-Modus und Clients können auf diesen zugreifen. Mithilfe des Dämons (siehe Abbildung 4.26**)** ist es möglich, Netzwerke auch über eine

Socket-Schnittstelle abzurufen und in ihnen zu navigieren sowie Daten abzubilden (siehe Kapitel 4.3). Man kann den Port wählen und hat die Möglichkeit, eine Zeit vorzugeben, nach der automatisch untätige Clients aussortiert werden.

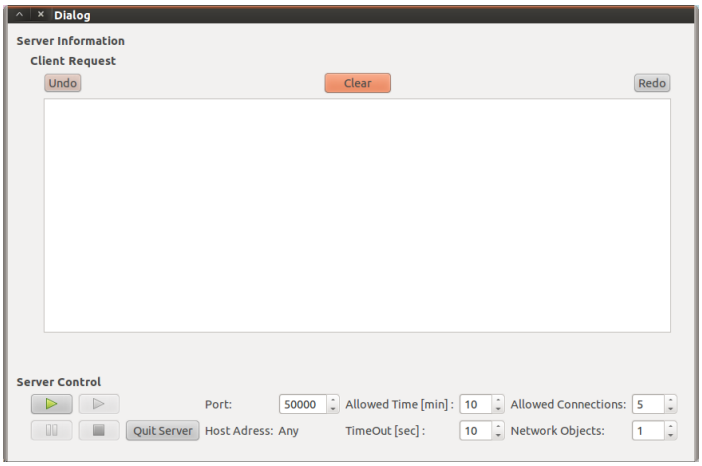

Abbildung 4.26: Dämon Bedienoberfläche: Das GUI-basierte Frontend (deutsch: vorderes Ende) bietet dem Nutzer verschiedene Konfigurationsmöglichkeiten. Der Dämon kann gestartet, pausiert sowie gestoppt werden. Auch lässt sich der Port einstellen. Ein integrierter Textbrowser ermöglicht zudem, die clientseitigen Anfragen chronologisch zu speichern und zu lesen.

Tool „Network Object Selector“

MapOmnia bietet die Möglichkeit, Teilnetzwerke z. B. durch Selektion im „Network Visualisation Fenster“ zu erstellen. Möchte der Benutzer ein neues Teilnetzwerk durch die Auswahl aus anderen Teilnetzwerken bzw. des Gesamtnetzwerks erstellen, so kann dieses dialogbasiert durch Ausführen dieses Tools geschehen. Es öffnet sich ein Dialogfenster (siehe Abbildung 4.27). Ein Baum wird erstellt, welcher analog dem „Network Data Tree“ aufgebaut ist, nur mit dem Unterschied, dass nun an den Baumknoten zusätzlich Checkboxen (deutsch: das Markierungsfeld) angeheftet sind. Während der Benutzer durch den Baum navigiert, wird das „Network Visualisation Fenster“ interaktiv aktualisiert. Der Benutzer kann an diejenige Komponenten Häkchen setzen, die er in dem neuen Teilnetzwerk darstellen lassen möchte. Insbesondere für Modellierer für Organismen kann dieses ein hilfreiches Tool darstellen. In dem Anwendungsbeispiel „Konstruktion eines neuen Teilnetzwerkes“ wird dieses exemplarisch gezeigt.

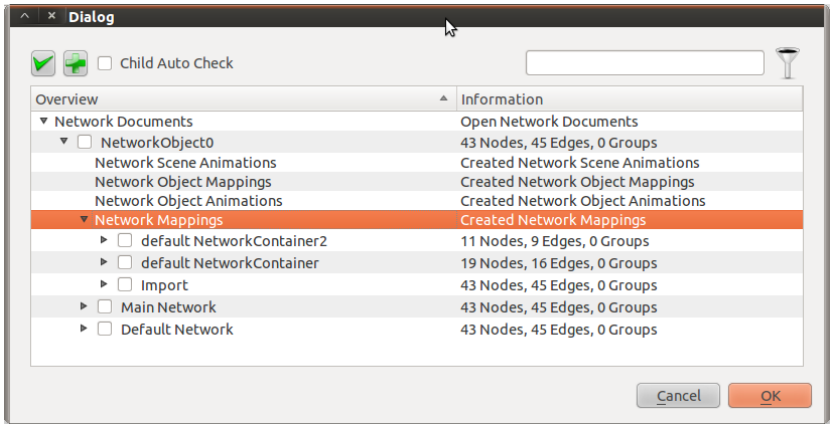

Abbildung 4.27: Das Tool „Network Object Selector“ bietet die Möglichkeit, neue Teilnetzwerke zu kreieren. Hierbei kann die Checkbox ein und ausgestellt werden, um die Elemente zu markieren, die in dem zu erstellenden Netzwerk enthalten sein sollen.

Tool „Shortest Path“

Dieses Tool ermöglicht es, die kürzesten Wege zwischen ausgewählten Knoten zu berechnen. Hierfür wird der Dijkstra-Algorithmus (vergleiche Kapitel 2.1.4.3), welcher in der C++ Bibliothek Boost (siehe Kapitel 3.7.5) bereitgestellt wird, verwendet. Es wird Undo-basiert ein neues Teilnetzwerk erstellt, das sämtliche gefundenen Wege, als Gruppen gespeichert, enthält. Jede Gruppe wird hierbei in den Render-Modus gesetzt, welcher vorsieht, den Weg grafisch zu unterlegen. Dadurch wird der Weg schlangenartig hinterlegt. Dieses Darstellungsverhalten ist exemplarisch in Abbildung 4.28 gezeigt.

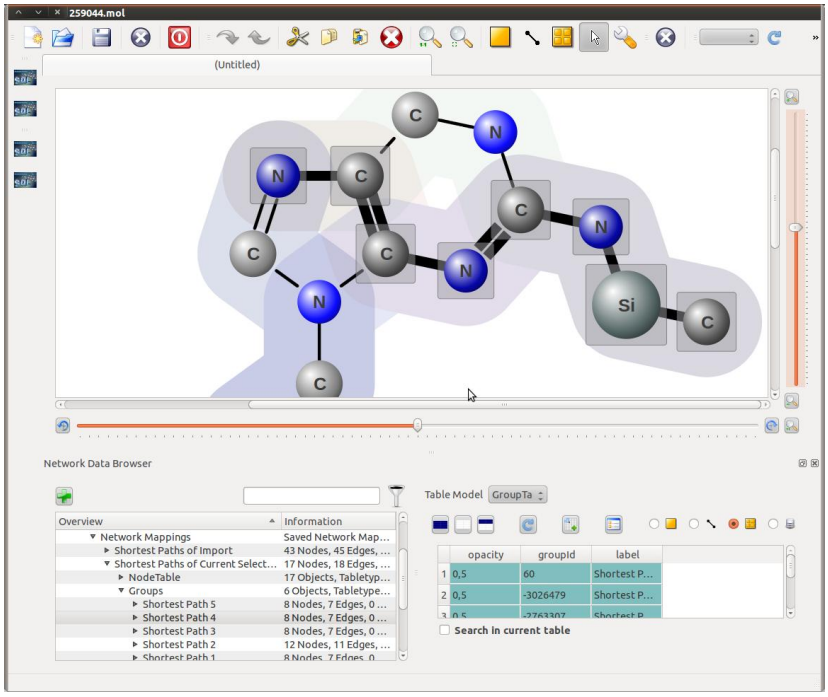

Abbildung 4.28: Die Shortest Path Funktion des Programms berechnet sämtliche kürzesten Wege mittels des Dijkstra Algorithmus. Es wird ein Teilnetzwerk erstellt, in dem die kürzesten Wege als Gruppen gespeichert werden. Wählt der Benutzer eine Gruppe aus, so wird der aktuelle Weg markiert, wobei die Knoten mit grauem Kästchen unterlegt und die Kanten fett dargestellt werden.

Tool „Identify Enzymes And Substances“

Wurde ein metabolisches Netzwerk geladen, das keine Informationen beinhaltet, ob es sich bei den Knoten um Enzyme oder Substanzen handelt, kann durch Ausführen dieses Tools, diese Information den Knoten zugeordnet werden. Die Zuordnung erfolgt über eine Instanz des *BRENDADatabaseObject* (beschrieben in Kapitel 4.1.4.6).

Tool „Adjust Edge Length“

Hat der Benutzer Kanten selektiert, kann er die Kantenlänge anpassen. Für diesen Zweck gibt es das Tool „Adjust Edge Length“. Es öffnet sich ein Dialogfenster, in welchem automatisch die Durchschnittslänge der gewählten Kanten eingetragen wird. Nun kann eine andere Länge vorgegeben werden. Iterativ wird dann versucht, die Kantenlänge neu zu setzen. Bei jeder Kante wird ausgehend von ihrem Anfangsknoten, der Endknoten auf

die gewünschte Länge verschoben, wobei die Winkellage in der Ebene nicht verändert wird. Die jeweilige Veränderung der Kantenlängen haben ihrerseits Auswirkungen auf die im Netzwerk befindlichen übrigen Kanten. Zu diesem Zweck optimiert der Algorithmus so lange, bis das gewünschte Ziel (Gleichheit der Länge aller selektierten Kanten) oder ein Iterationswert von 100000 erreicht wird. In Abbildung 4.29 wurde die Anpassung der Kantenlänge exemplarisch für das Molekül 2'-Deoxyguanosine_5'-p_5TMS durchgeführt.

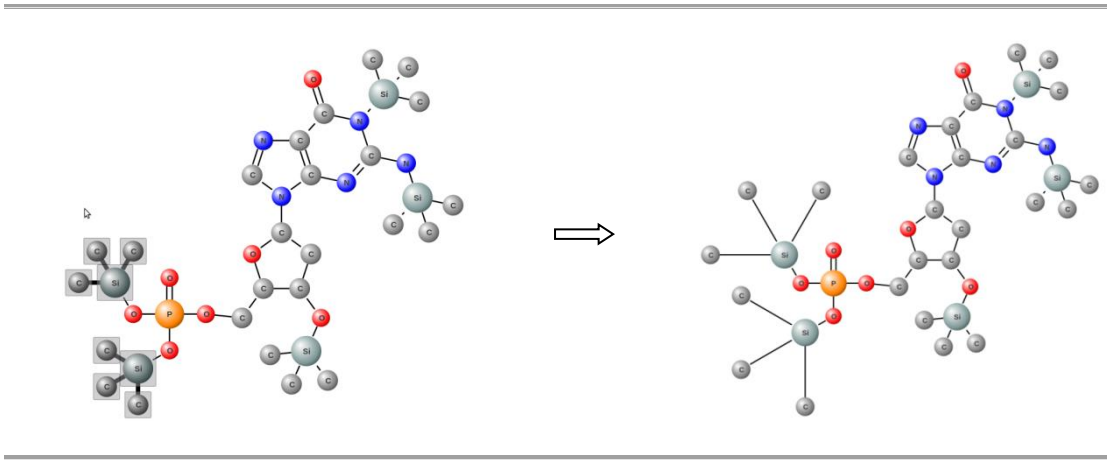

Abbildung 4.29: In dem Molekül 2'-Deoxyguanosine_5'-p_5TMS wurden sechs Kanten ausgewählt. Durch Verwendung des Tools „Adjust Edge Length“ sind die Kantenlängen neu definiert worden und es ergibt sich das rechte Bild.

Tool „Consistency Check Edges“

Teilnetzwerke können Kanten enthalten, dessen Knoten nicht in dem Teilnetzwerk enthalten sind. Um das Entfernen dieser Kanten zu ermöglichen, wird das Tool „Consistency Check Edges“ angeboten.

Beispiel:

In Abbildung 4.30 wird das Molekül 2'-Deoxyguanosine_5'-p_5TMS gezeigt, in dem eine Selektion vorliegt. Wird nun aus dieser Selektion ein Teilnetzwerk erstellt, so ergibt sich zunächst das in Abbildung 4.31 (linkes Bild) gezeigte Zwischenergebnis. Führt man anschließend das Tool „Consistency Check Edges“ aus, so werden diejenigen Kanten entfernt, deren Knoten nicht im Teilnetzwerk vorhanden sind (rechtes Bild). Die Entfernung der Kanten ist Undo-basiert.

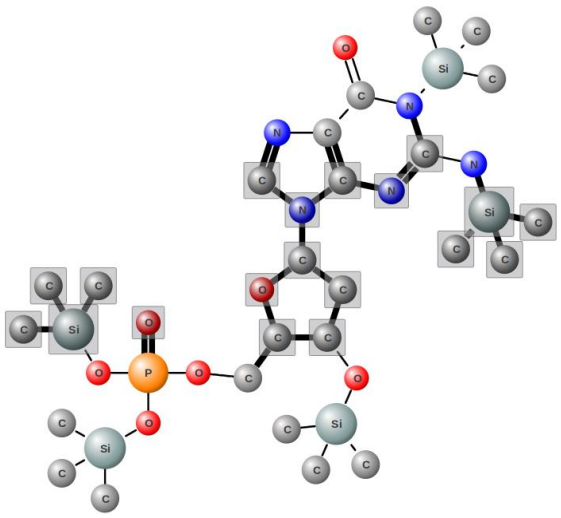

Abbildung 4.30: Das Molekül 2'-Deoxyguanosine_5'-p_5TMS mit selektierten Knoten (hier: Atome) und Kanten (hier: chemische Bindungen).

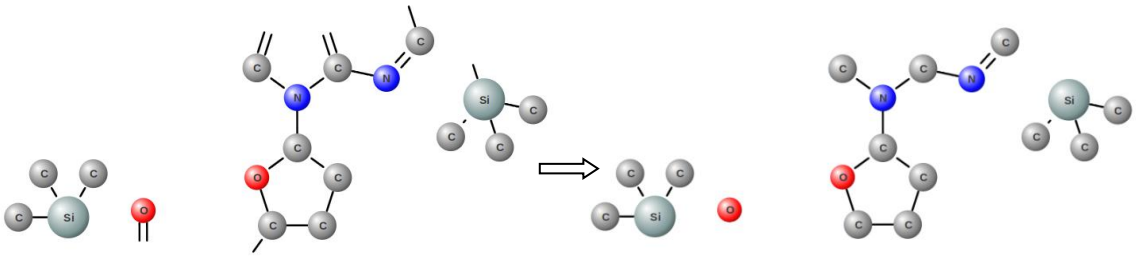

Abbildung 4.31: Links ist das Teilnetzwerk dargestellt, das aus der Selektion von Abbildung 4.30 entsteht. Rechts wurden die Kanten (hier: chemische Bindungen) entfernt, zu denen keine Knoten im Teilnetzwerk existieren.

Tool „Add Missing Edges“

Nach Auswahl eines Teilnetzwerks kann der Benutzer durch Ausführen dieses Tools Kanten automatisch zu Knoten hinzufügen lassen, welche in diesem Netzwerk im Vergleich zu dem Gesamtnetzwerk fehlen. Diese Addition von Kanten erfolgt Undo-basiert.

Beispiel:

In Abbildung 4.32 ist das Molekül 2'-Deoxyguanosine_5'-p_5TMS gezeigt, bei dem eine Selektion von Knoten (hier: Atome) vorliegt. Aus dieser Selektion wird, wie in Abbildung 4.33 gezeigt, ein Teilnetzwerk erstellt (linkes Bild). Nach Ausführen des Tools werden alle Kanten (hier: chemische Bindungen) Undo-basiert hinzugefügt, die im Gesamtnetzwerk vorhanden sind und in diesem Teilnetzwerk Start- und Endknoten besitzen (rechtes Bild).

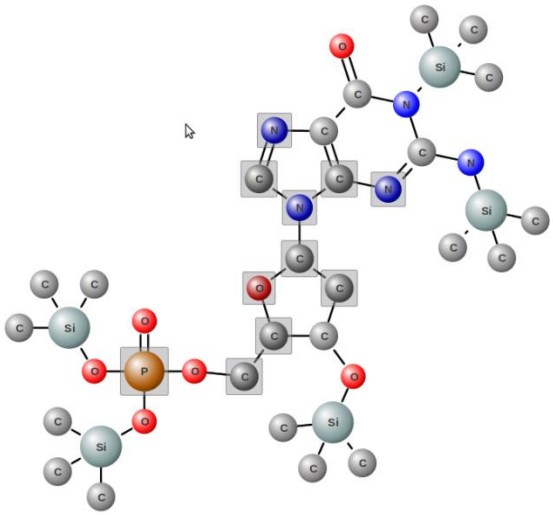

Abbildung 4.32: Das 2'-Deoxyguanosine_5'-p_5TMS Molekül mit selektierten Knoten (hier: Atome).

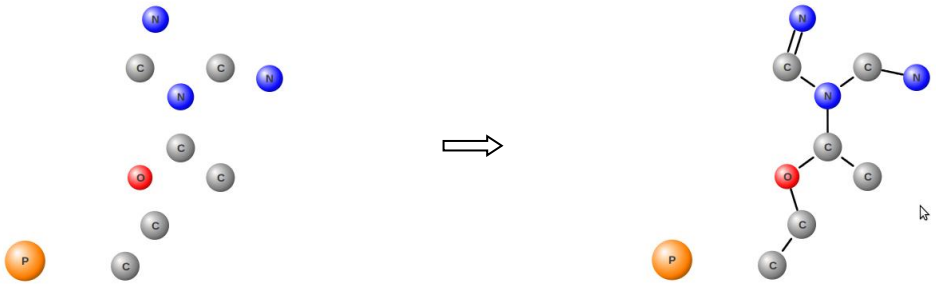

Abbildung 4.33: Links ist das Teilnetzwerk gezeigt, welches aus den selektierten Atomen in Abbildung 4.32 entsteht. Auf der rechten Seite wurden die Kanten (hier: chemische Bindungen) hinzugefügt, die auch in dem Gesamtnetzwerk vorhanden sind und ebenfalls die Knoten aus dem Teilnetzwerk besitzen.

Tool „Create Selection From…“

Neben der Möglichkeit, dialogbasiert Teilnetzwerke, wie im Tool „Network Object Selector“ beschrieben, zu erstellen, bietet MapOmnia ebenfalls die Möglichkeit, Selektionen auf das aktuell ausgewählte Netzwerk von anderen Teilnetzwerken zu erstellen. Dieses wird dialogbasiert ermöglicht. Hierbei wird ein Baum mit Teilnetzwerken sowie den dazugehörigen Netzwerkobjekten erstellt. Die Baumknoten verfügen zur Auswählbarkeit über Checkboxen. Während der Benutzer durch den Baum navigiert, wird das „Network Visualisation Fenster“ aktualisiert und auf die Auswahl des Baumes fokussiert. An den Komponenten können Häkchen gesetzt werden, die der Nutzer als Selektion im aktuellen Teilnetzwerk haben möchte.

Wurde beispielsweise die BRENDA-Maps inklusive verschiedener Organismen-Teilnetzwerke (vergleiche Anwendungsbeispiel in Kapitel 4.1.7.1) geladen, könnte sich der Benutzer z. B. alle Knoten (hier: Metabolite und Enzyme) von Organismus x markieren lassen, die auch im Organismus y vorhanden sind. Hierdurch lassen sich die gemeinsamen Metaboliten und Enzyme über diese Funktion anzeigen.

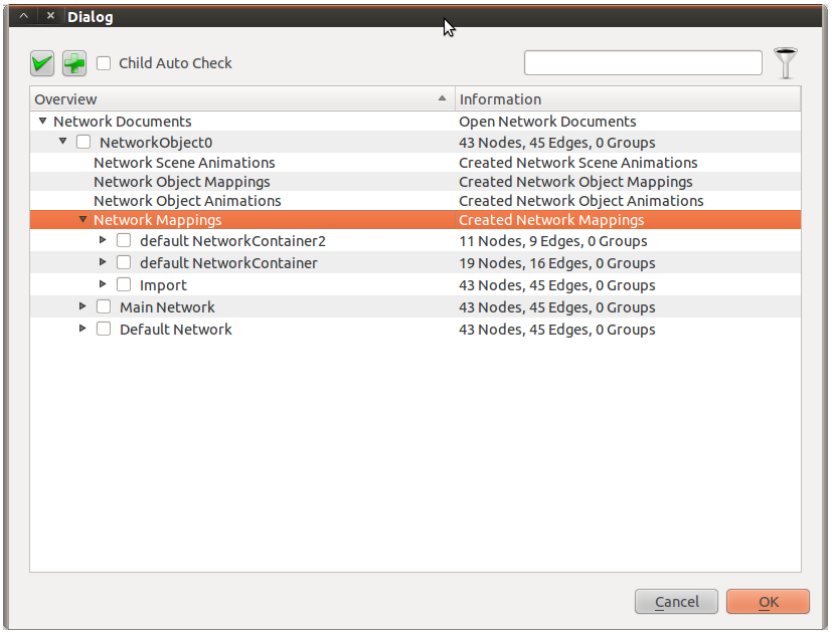

Abbildung 4.34: Das Tool „Create Selection From…“ bietet die Möglichkeit, neue Selektionen in einem ausgewählten Netzwerk zu erstellen. Die Auswahl erfolgt über angehängte Checkboxen.

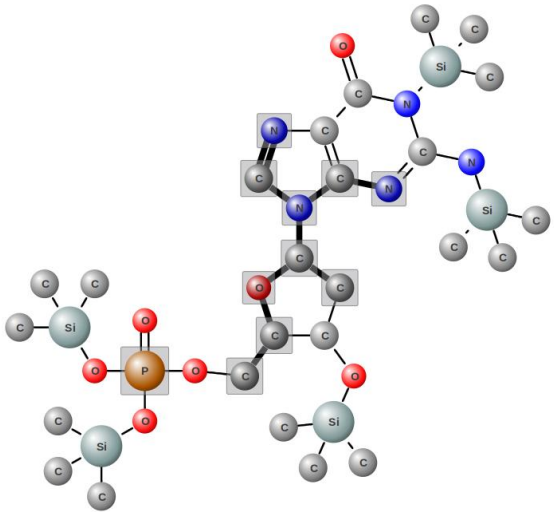

Abbildung 4.35: In dem obigen gezeigten Molekül wurden Kanten und Knoten mithilfe des Tools „Create Selection From…“ selektiert.

Beispiel:

Ausgehend von dem Molekül, dargestellt in Abbildung 4.32, wurde das Teilnetzwerk für die Selektion aus Abbildung 4.33 (rechte Seite) gewählt. Die Selektion erfolgt über Auswahlboxen (Abbildung 4.34). Das Selektionsergebnis ist in Abbildung 4.35 dargestellt.

Tool „Create Reaction Groups“

Handelt es sich bei einem geladenen Netzwerk um ein metabolisches Netz und sind Informationen vorhanden, welche der Knoten die Enzyme repräsentieren, so können Reaktionsgruppen erzeugt werden, welche die Reaktionen des Netzwerkes beinhalten. Die Ergänzung der Gruppen erfolgt Undo-basiert.

Tool „Extend to Neighbours“

Dieses Tool erlaubt dem Anwender, innerhalb eines Teilnetzes dieses Netzwerk in Referenz zu dem Gesamtnetzwerk sukzessive zu erweitern. Hierbei werden an sämtliche Knoten der nächsten Nachbarn, inklusive der dazugehörigen Kanten, zu dem ausgewähltem Teilnetzwerk Knoten und Kanten hinzugefügt.

Beispiel:

Ausgehend vom metabolischen Netz für den Organismus *Acinetobacter baumannii UMB003* (Abbildung 4.36 oben), als einem Teilnetzwerk der BRENDA-Maps (siehe Kapitel 3.9.5) wird das Tool zweimalig ausgeführt (Abbildung 4.36). Der Organismus stammt hierbei aus der PATRIC-basierten Annotation (siehe Kapitel 3.9.4). Das neu entstandene Netzwerk enthält dann die um eins erweiterten Reaktionen (Abbildung 4.36 unten) im Vergleich zum Ursprungsnetzwerk.

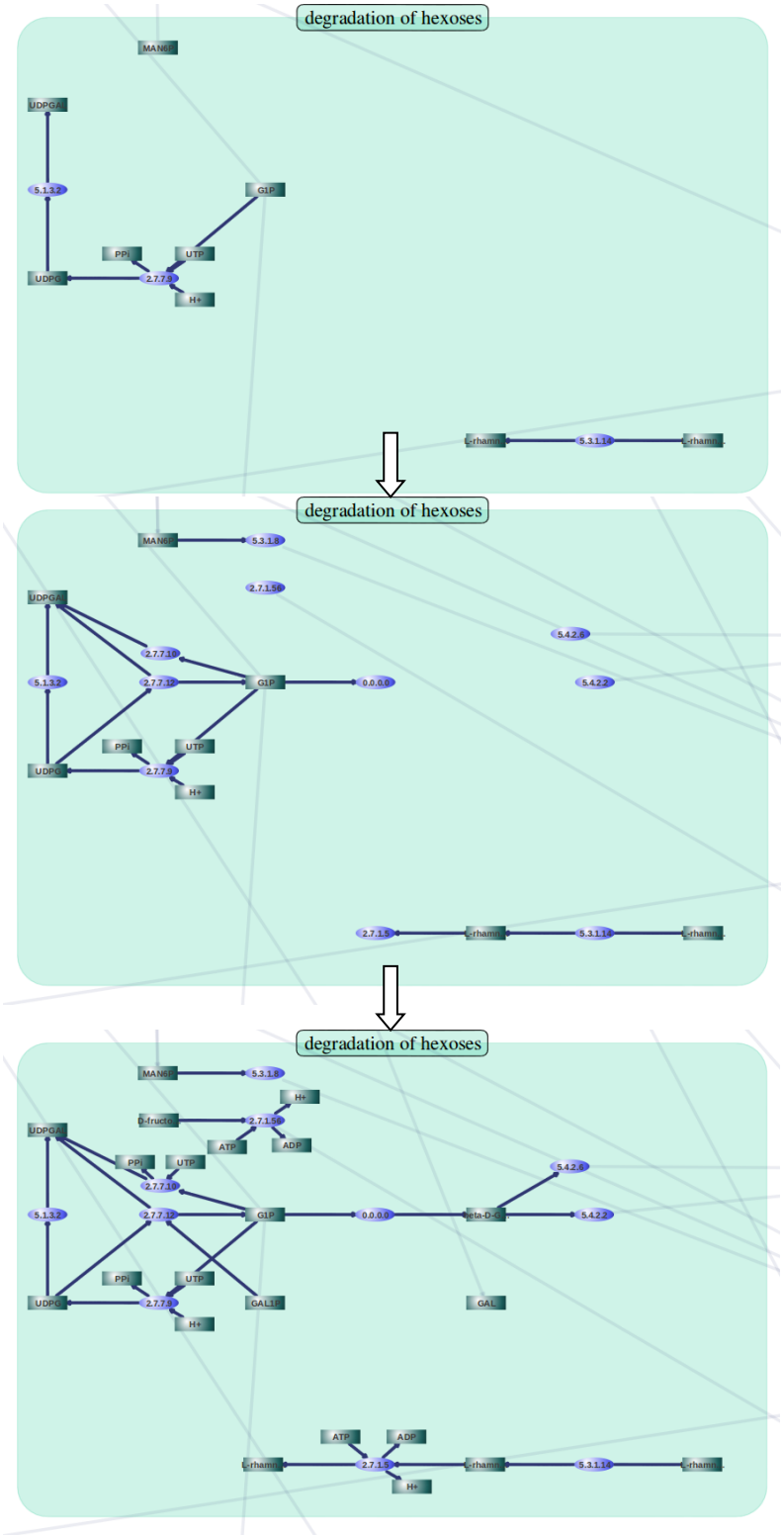

Abbildung 4.36: *Acinetobacter baumannii UMB003* wird als Ausgangsteilnetzwerk sukzessive um zwei Nachbarn zur Referenz des Gesamtnetzwerks erweitert.

Tool „Create Groups From Reference“

Durch Ausführen des Tools „Create Groups From Reference“ können für ein selektiertes Teilnetzwerk automatisiert Gruppen erstellt werden, die mit den Gruppen aus der Gesamtkarte verglichen werden. Hierbei kann beispielsweise eine Gruppe ein Stoffwechselweg oder eine Reaktion darstellen. Es ist ein Prozentwert vorzugeben, der festlegt, wie viele Knoten mindestens in dem aktuellen Teilnetz in Bezug auf eine bestimmte Gruppe vorhanden sein müssen, damit die Knoten der neu zu erstellenden Gruppe zugeordnet werden.

Beispiel:

Als Ausgang für das Tool dient eine organismusspezifische Karte von *Corynebacterium glutamicum ATCC 13032* (Abbildung 4.37 oben), als Teilnetzwerk der BRENDA-Maps (siehe Kapitel 3.9.5). Durch Ausführen des Tools mit 60 % als Vorgabewert erhält man das in Abbildung 4.37 unten dargestellte Bild. Der Organismus stammt hierbei aus der BRENDA-basierten Annotation. Das neu entstandene Teilnetzwerk enthält nun alle Gruppen des Gesamtnetzwerkes, die innerhalb des Teilnetzwerks mindestens 60% Knoten besitzen. Abbildung 4.38 oben und unten zeigen hierbei einen vergrößerten Ausschnitt der jeweiligen Teilnetzwerke.

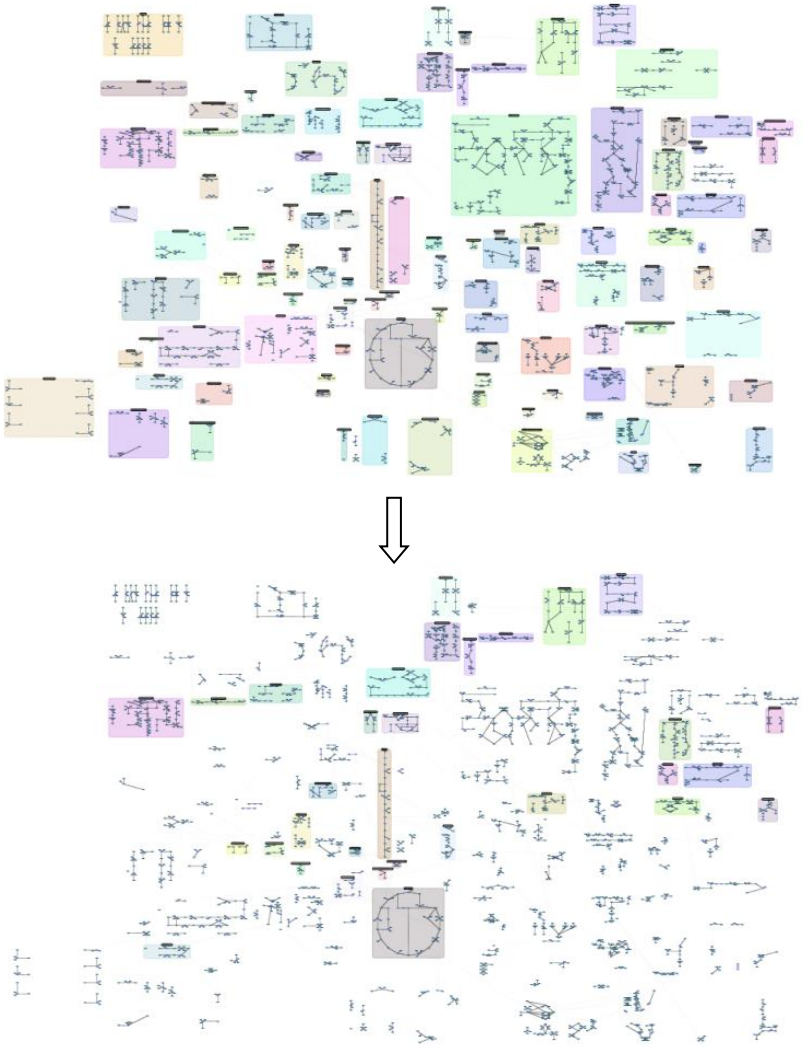

Abbildung 4.37: *Corynebacterium glutamicum ATCC 13032* (oben) wird als Ausgangsteilnetzwerk auf Vorhandensein von Knoten (hier: Enzyme und Metabolite) pro Stoffwechselweg analysiert. Sofern mindestens 60% der jeweiligen Knoten pro Stoffwechselweg vorhanden sind, wird der Stoffwechselweg dem neu erstellten Teilnetzwerk hinzugefügt (unten). Es ist das gesamte Netzwerk abgebildet.

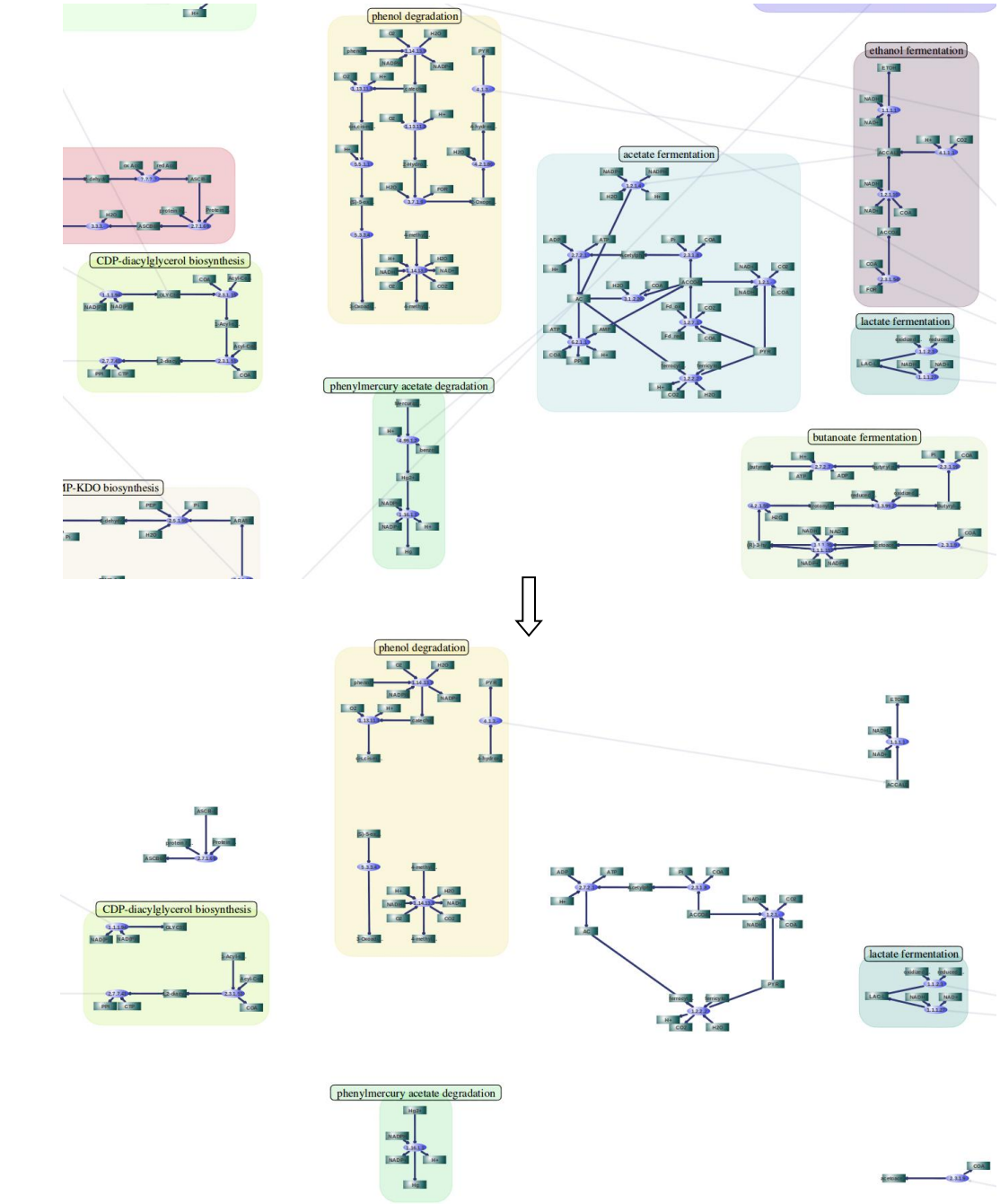

Abbildung 4.38: *Corynebacterium glutamicum ATCC 13032* (oben) wird als Ausgangsteilnetzwerk auf Vorhandensein von Knoten (hier: Enzyme und Metabolite) pro Stoffwechselweg analysiert. Sofern mindestens 60% der jeweiligen Knoten pro Stoffwechselweg vorhanden sind, wird der Stoffwechselweg dem neu erstellten Teilnetzwerk hinzugefügt (unten). Es ist ein Ausschnitt abgebildet.

4.1.5.14 Wizards

Werden komplexere Arbeitsabläufe für die Einstellung bestimmter Programmfeatures erforderlich, so bietet es sich an, Wizards (deutsch: der Assistent) zu implementieren. Wizards sind eine besondere Art von Eingabe-Dialogen, die eine Abfolge von Dialogfenstern beinhalteten. Der Benutzer wird Schritt für Schritt durch den Dialog gesteuert. In MapOmnia sind momentan Wizards zugänglich, welche die Durchführung von Datenabbildungen auf Netzwerke erleichtern.

In Abbildung 4.39 ist die Reihenfolge der Dialogfenster beispielhaft für das Ausführen einer Farbcodierung gezeigt. Ausgewählte Einstellungen lassen sich jederzeit rückgängig machen, indem man zwischen den Fenstern vor und zurück navigiert. Neben der Farbcodierung können auch die Größencodierung und Anpassungen der Kantenstärke per Wizard ausgewählt werden. Der Ablauf ist analog. Eine Erstellung der Codierungs-Varianten ist im Anwendungsbeispiel „Experimentdaten visualisieren“ im Kapitel 4.1.7.3 exemplarisch aufgezeigt.

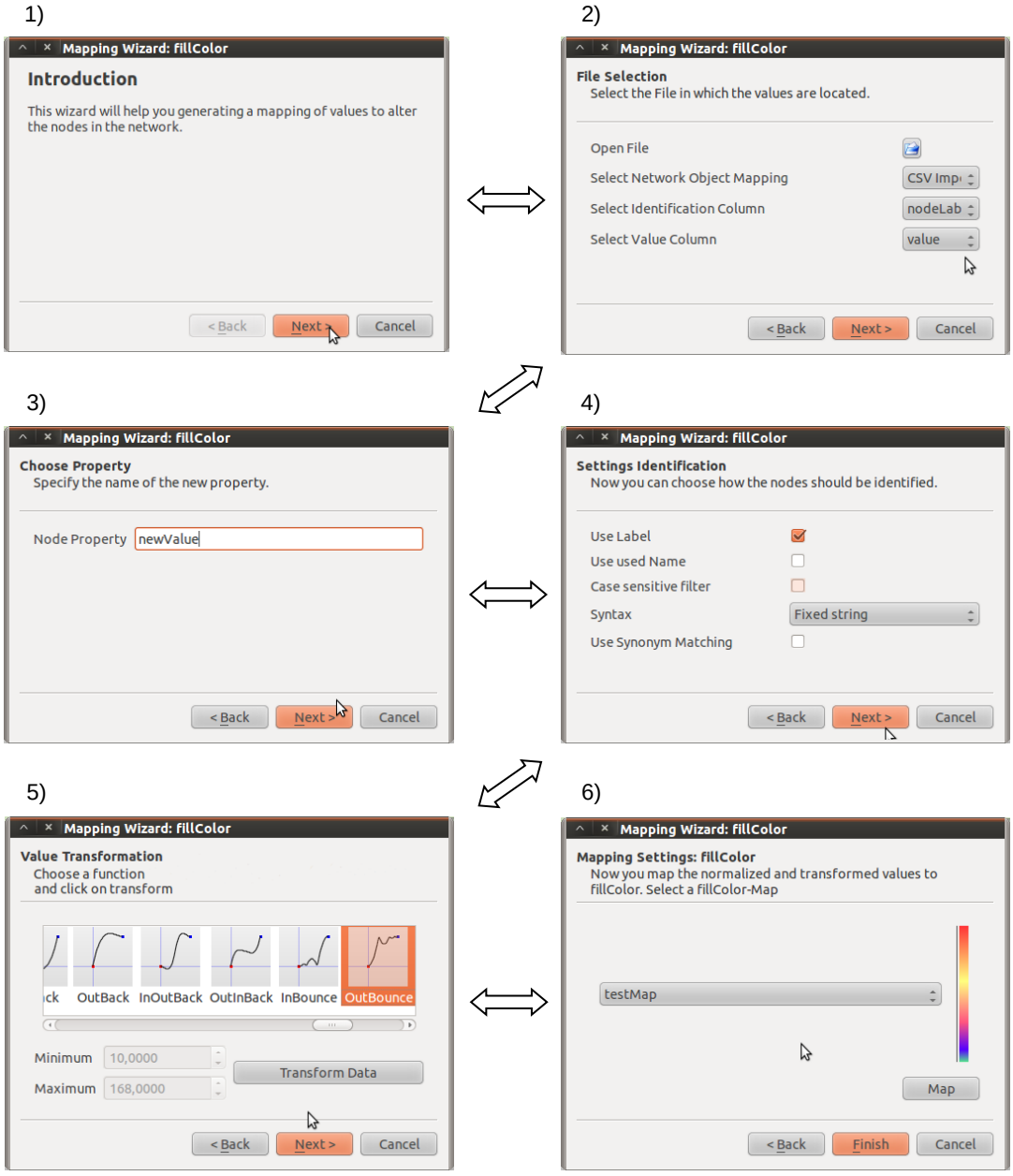

Abbildung 4.39: Dialoggeführtes Erstellen einer Farbcodierung mit Hilfe des Wizards. Das erste Fenster bietet eine kleine Einleitung. Im zweiten Fenster wird der Datei-basierte Import durchgeführt. Das dritte dient der Benennung des Attributes, im vierten werden die Identifikationseinstellungen übernommen. Transformationen der eingelesenen Werte können im fünften Fenster eingestellt werden und im letzten Fenster wird schließlich die Colormap gewählt und die Codierung durchgeführt.

4.1.5.15 Menüleiste

Die Menüleiste beinhaltet die Menüpunkte des Programms und dient als Bedieninterface für den Nutzer von MapOmnia. Sie ist in folgende Hauptmenüs eingeteilt:

- File
- Edit
- Create
- Modify
- Update
- Tools
- Wizards
- Online Resources
- Window
- Plugins
- Script
- Settings
- Help

Unterhalb dieser Menüpunkte sind weitere Bedienelemente lokalisiert. Der Benutzer kann Dateien öffnen und speichern, die Bearbeitung des geöffneten Netzwerkes vornehmen, Tools und Wizards benutzen und auf diverse Einstellmöglichkeiten zugreifen. Um die Benutzung des Programmes zu erleichtern, wurden die meisten Programmfunktionen mit Shortcut-Funktionalität belegt. Die zentralen Elemente des Menüs werden im Folgenden beschrieben.

Menüpunkt „File“

In dem Menü „File**“** lassen sich Netzwerk-Dateien öffnen, neue Netzwerke erstellen und abspeichern. Das Menü verfügt über ein „Recent Networks“ Untermenü, welches die bisher geöffneten Dateien verwaltet. Das Programm verfügt über eine komplexe *IOHandler*-Klasse (siehe Kapitel 4.1.4.7), die den Import sowie Export verwaltet. Es

werden die Formate XGMML (Kapitel 3.8.1), KGML (Kapitel 3.8.2), SBML (Kapitel 3.8.3), GML (Kapitel 3.8.4), CSV (Kapitel 3.8.5), Molfiles (Kapitel 3.8.6) sowie ein binäres Format (Kapitel 3.8.8) für den Import von Netzwerkdaten unterstützt. Der Export kann im GML, in einem Binär- sowie CSV-Format erfolgen. Die Erkennung des Dateiformates beim Import erfolgt automatisch, durch eine Funktion, die überprüft welches Dateiformat vorliegt und beim Export durch Berücksichtigung des Dateisuffixes.

Des Weiteren verfügt das Programm über eine Datenbankschnittstelle, um Netzwerke direkt aus einer Datenbank auszulesen. Durch die von Qt bereitgestellten Datenbank Treiber werden die wichtigsten Datenbanktypen, wie Borland InterBase, MySQL, Oracle Call Interface Driver, Open Database Connectivity (ODBC), Microsoft SQL Server, PostgreSQL, SQLite und Sybase Adaptive Server unterstützt.

Möchte man mehrere Dateien des Typs Text, Bild oder die Verlinkung zu Dateienpfaden laden, so kann dieses über das Untermenü „Import“ geschehen. Es steht ein Export für Bild-Dateien, wie PNG, JPG usw. (siehe Tabelle 4.7) sowie SVG und PDF zur Verfügung.

Tabelle 4.7: Die unterstützten grafischen Exportformate (entnommen aus der Qt Hilfe der Qt Dokumentation; Nokia Corporation, 2013a)

Format	Beschreibung	Qt's Support
BMP	Windows Bitmap	Lesen/Schreiben
GIF	Graphic Interchange Format (optional)	Lesen
JPG	Joint Photographic Experts Group	Lesen/Schreiben
JPEG	Joint Photographic Experts Group	Lesen/Schreiben
PNG	Portable Network Graphics	Lesen/Schreiben
PBM	Portable Bitmap	Lesen
PGM	Portable Graymap	Lesen
PPM	Portable Pixmap	Lesen/Schreiben
TIFF	Tagged Image File Format	Lesen/Schreiben
XBM	X11 Bitmap	Lesen/Schreiben
XPM	X11 Pixmap	Lesen/Schreiben

Menüpunkt „Edit“

Das Menü „Edit“ stellt Netzwerkmanipulationen wie Ausschneiden, Einfügen, Löschen und Kopieren zur Verfügung Abbildung 4.40 stellt exemplarisch dar, wie sich ausgewählte Bereiche kopieren lassen. Ebenso lassen sich durch eine implementierte Undo/Redo-Funktionalität (siehe Kapitel 4.1.4.5) Operationen auf dem Netzwerk rückgängig machen. Dies kann außerdem über ein einem Docking Fenster erfolgen, welches die Operationen in einem Undo-Stack bereitstellt.

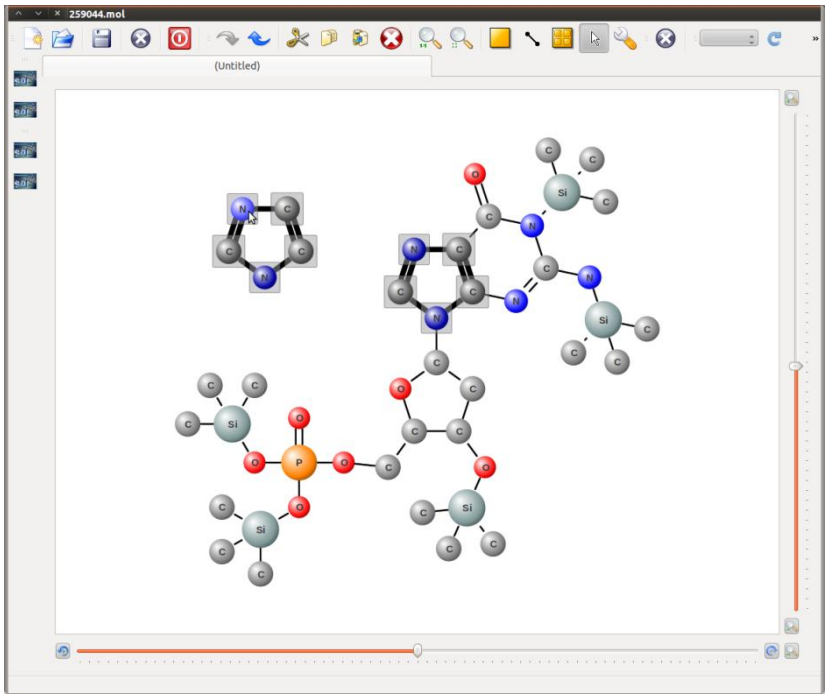

Abbildung 4.40: Beispiel für eine „Copy and Paste“ Aktion in MapOmnia. Der im Molekül enthaltene 5er-Ring wurde markiert und kopiert. Durch Gedrückthalten der Strg-Taste wird die Selektion der der ausgewählten Substruktur für weitere Einfüge-Operationen gespeichert.

Die Selektion aller Netzwerkobjekte (Knoten, Kanten und Gruppen) der gesamten Karte oder eines Bereichs lässt sich in diesem Menü durchführen. Man kann ebenso eine vorhandene Selektion für alle Netzwerkobjekte oder nur eine bestimmte Art von Netzwerkobjekten (z. B. nur Kanten) umschalten (siehe Abbildung 4.42). Eine erweiterte Selektionsmöglichkeit ist die „Expand Selection“ (siehe Abbildung 4.41) und

„Decrease Selection“-Option. Hier ist eine Expansion oder Reduktion der Selektion bezüglich des Netzwerkes möglich.

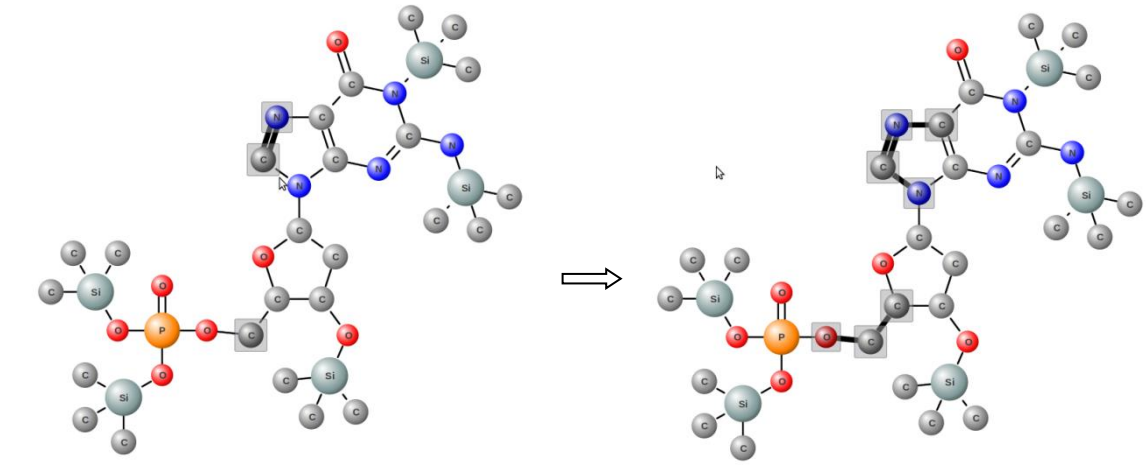

Abbildung 4.41: Beispiel für eine „Extend Selection“-Aktion in MapOmnia. Die Selektion wird entlang der Kanten vergrößert.

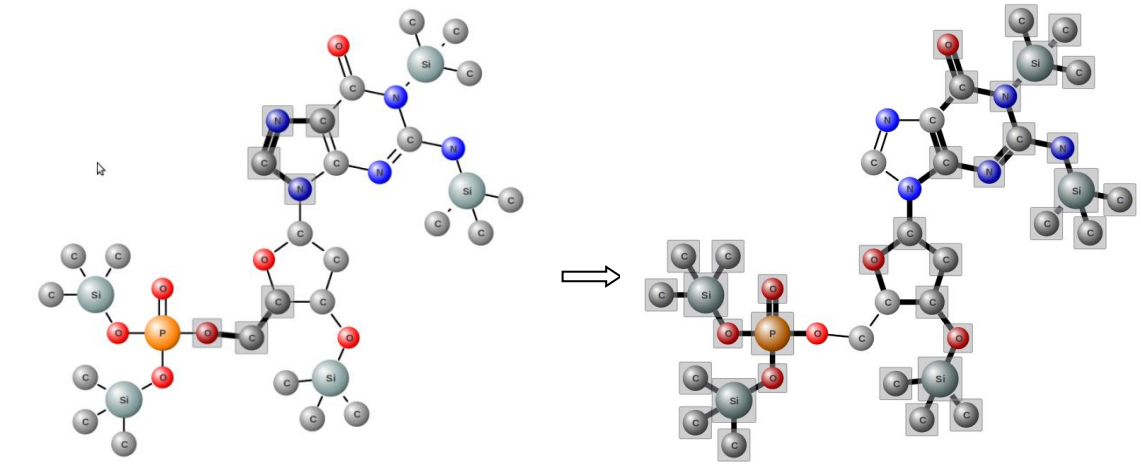

Abbildung 4.42: Beispiel für eine „Toggle“-Markierung in MapOmnia. Es werden sämtliche Knoten und Kanten des Netzwerkes selektiert, welche zuvor nicht selektiert waren. Die zuvor selektierten Objekte sind nicht mehr selektiert.

Möchte man aus der vorhandenen Selektion eine Gruppe, ein Teilnetzwerk oder ein neues Netzwerk (als neuen Reiter) erstellen, so kann dieses im Untermenü „Selection Create“ erfolgen. Soll zu den selektierten Netzwerkobjekten eine neue Eigenschaft hinzugefügt werden, ist dieses durch „Add Property“ möglich. Hierbei ist der gewünschte Datentyp auszuwählen und es öffnet sich das in Abbildung 4.43 dargestellte Fenster. Man kann hierbei zwischen den Datentypen wählen, die von *QVariant* unterstützt werden (Tabelle A.1 Anhang).

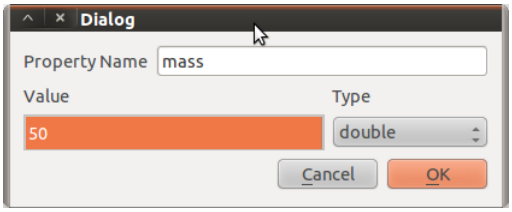

Abbildung 4.43: Das Dialogfenster ermöglicht die dynamische Addition von Attributen zu den Netzwerkobjekten. Es werden hierbei ein Attributwert sowie der Datentyp gewählt.

Menüpunkt „Create“

Der Menüpunkt „Create“ ermöglicht dem Benutzer, Styles für die Netzwerkobjekte sowie Color- und Sizemaps für die Abbildung von Daten dialogbasiert zu erstellen. Auch das Erstellen von Animationen, was in Kapitel 4.1.5.7 und 4.1.5.9 genauer beschrieben wird, wird unter diesem Menüpunkt bereitgestellt.

In Abbildung 4.44 ist das Dialogfenster für die Erstellung eines Knotenstyles exemplarisch gezeigt. Es können Einstellungen vorgenommen werden, die für die grafische Repräsentation des Netzwerkobjekts wichtig sind. Möchte man einen Style speichern, so unterstützt MapOmnia sowohl Ini- als auch ein Binär-Format. Die Dateien werden in einem Unterordner des Programms abgelegt.

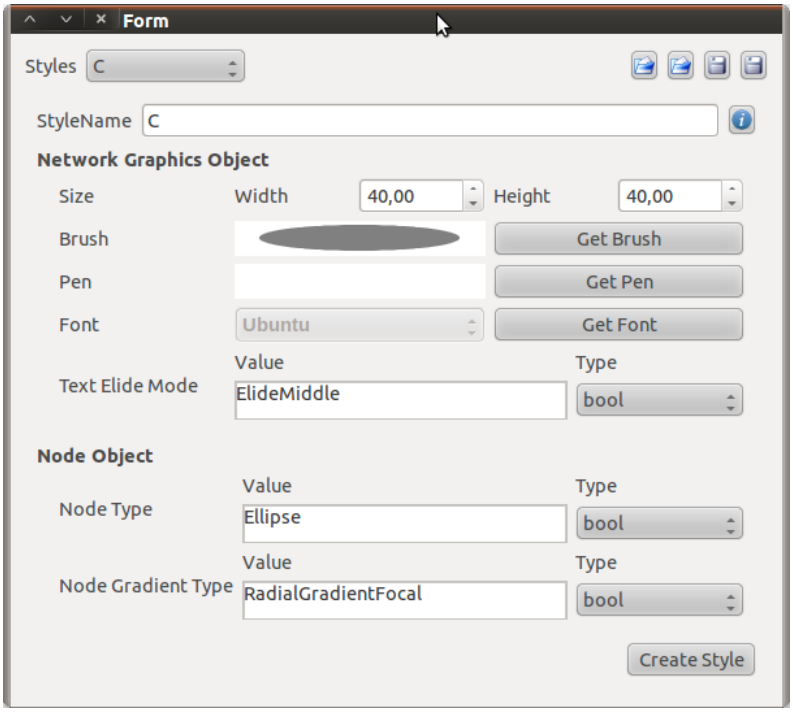

Abbildung 4.44: „Create Style Dialog“: In diesem Fenster sind sämtliche grafisch relevanten Einstellmöglichkeiten für ein Netzwerkobjekt (in diesem Fall für Knoten) zusammengefasst. Erstellte Styles lassen sich im Ini- und Binär-Format speichern.

Abbildung 4.45 zeigt ein Dialogfenster, mit dem eine Colormap erstellt werden kann. Es werden Werte zwischen 0 und 1 Farben zugeordnet. Die Farbauswahl erfolgt über ein eigenes Dialogfenster. Neben der Colormap lässt sich eine Sizemap auf die gleiche Weise erstellen, wobei hier den Zahl-Werten zwischen 0 und 1 Rechtecken zugeordnet werden.

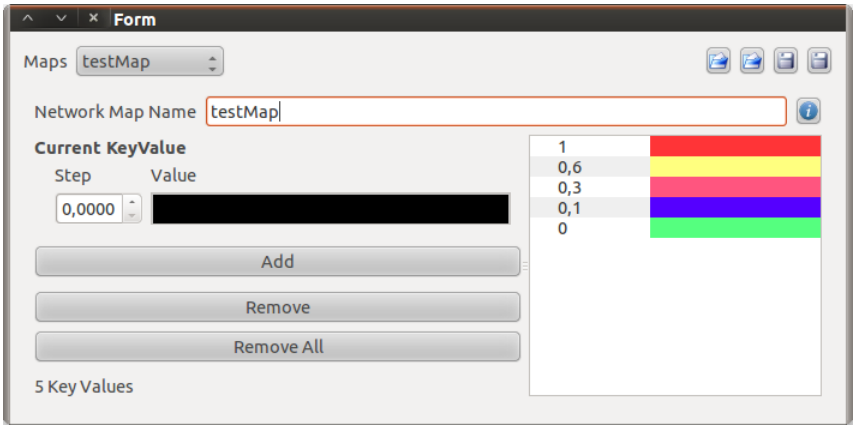

Abbildung 4.45: „Create Map“-Dialog Fenster: In diesem Fenster kann der Benutzer eine Colormap erstellen. Hierzu wird einem Wert zwischen 0 und 1 stellvertretend einer Farbe zugeordnet. Erstellte Maps lassen sich im Ini- und Binär-Format speichern.

Menüpunkt „Modify“

Modifikationsmöglichkeiten wie Transformationen, die Anwendung von Styles oder Layout-Implementierungen sind unter dem Menüpunkt „Modify“ zugänglich.

Transformationen werden hierbei technisch durch eine Transformationsmatrix auf die jeweils selektierten Knoten durchgeführt. Sie können rotiert sowie in x- und y-Richtung skaliert werden (Abbildung 4.46). Der Benutzer kann so generierte Positionsmaps speichern, um verschiedene Transformationsergebnisse direkt vergleichen zu können Abbildung 4.46 zeigt das Dialogfenster, über das die Parameter einer Transformation eingestellt werden.

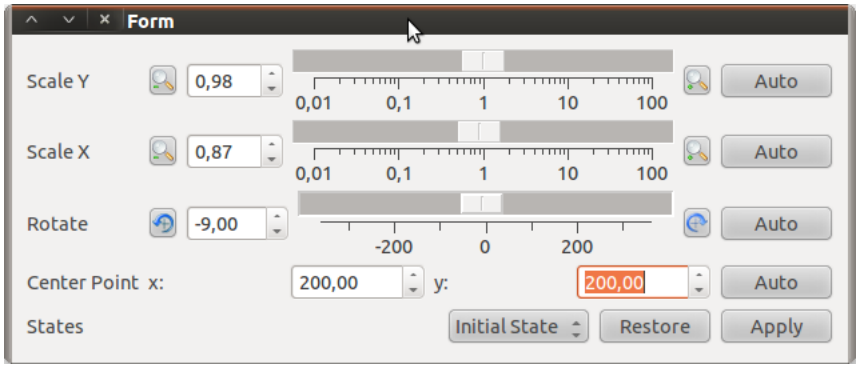

Abbildung 4.46: Transformationsmöglichkeiten von MapOmnia:. Ausgewählte Knoten lassen sich über dieses Fenster frei transformieren. Die Transformation kann in x- und y-Richtung erfolgen. Auch die Rotation um einen Mittelpunkt (englisch: Center-Point) wird ermöglicht. Die Speicherung der Positionsmaps wird ebenfalls unterstützt.

Sofern Styles für Netzwerkobjekte definiert wurden, können diese auch auf die selektierten Objekte angewendet werden. Abbildung 4.47 zeigt ein Dialogfenster für die Auswahl eines Styles für Knoten.

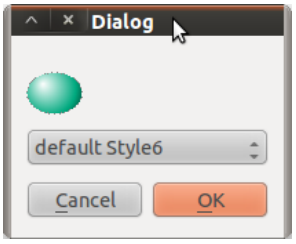

Abbildung 4.47: Zuvor erstellte Styles lassen sich einfach auf selektierte Netzwerkobjekte anwenden. Hierbei wird durch eine kleine Vorschau der gewählte Style visualisiert.

Im Menü „Modify“ sind folgende Layouts aufrufbar: „Linear Layout“ (deutsch: Lineare Anordnung), „Grid Layout“ (deutsch: Rasterbasierte Anordnung), „Elliptical Layout“ (deutsch: Elliptische Anordnung), „Painter Path Layout“ (deutsch: Anordnung entlang eine Zeichnungsweges) sowie „Force-Based Layout“ (deutsch: Kräfte- basierte Anordnung). Eine detaillierte Beschreibung der Layout-Möglichkeiten ist im Kapitel 4.1.6 vorgestellt.

Für die Z-Orientierung der Netzwerkobjekte, also Überlappungsverhalten, können die Objekte durch Anklicken von „Bring To Front“, „Send To Back“, „Bring Forwards“ sowie „Send Backwards“ in ihrer Ebene angepasst werden.
Der Menüpunkt „Mark“ gibt dem Benutzer die Möglichkeit, selektierte Netzwerkobjekte schnell zu verändern. Hierbei kann die grafische Erscheinung seines Brushs für die Objektfläche sowie die seines Brushs für die grafische Hervorhebung (englisch: highlighting) (siehe Abbildung 4.48) angepasst werden.

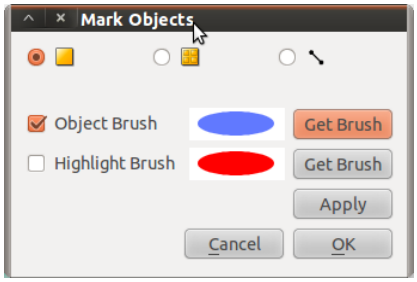

Abbildung 4.48: Durch das Dialogfenster „Mark Objects“ kann das Erscheinungsbild der ausgewählten Selektion an Knoten, Kanten und Gruppen schnell angepasst werden.

Menüpunkt „Update“

Manchmal kommt es vor, dass die implementierte Auto-Update-Funktion von verschiedenen Bereichen nicht richtig greift und es z. B. zu Artefakten in Darstellungen innerhalb des Netzwerkes im „Network Visualisation Fenster“ oder zu einer nicht vollständigen Baumstruktur im „Network Data Browser“ kommt. Zu diesem Zweck kann der Benutzer bei eventuell vorkommendem Fehlverhalten ein manuelles Update durchführen.

Menüpunkt „Tools“

Unter diesem Menüpunkt befinden sich die in MapOmnia implementierten Tools. Eine detaillierte Beschreibung der verfügbaren Tools ist in Kapitel 4.1.5.13 vorgestellt.

Menüpunkt „Wizards“

Unter diesem Menüpunkt befinden sich die für die Farb-, Größen- und Fluss-basierte Codierung verantwortlichen Wizards. Sie sind im Kapitel 4.1.5.14 ausführlich beschrieben.

Menüpunkt „Online Resources“

Wenn der Benutzer alternative Karten wie z.B. die BRENDA-Stoffwechselkarte (Kapitel 3.9.5) verwenden möchte, kann dieses unter dem Menüpunkt „Online Resources“ erfolgen. Es öffnet sich das in Abbildung 4.49 dargestellte Dialogfenster, über das er die gewünschten Organismen auswählen kann.

Neben der Möglichkeit die generische BRENDA-Stoffwechselkarte zu laden, können auch Organismus-spezifische Teilnetzwerke aus BRENDA sowie PATRIC (Kapitel 3.9.4) geladen werden. In dem Anwendungsbeispiel „Konstruktion eines neuen Teilnetzwerkes“ in Kapitel 4.1.7.1 wird exemplarisch die Verwendung von „Online Resources“ gezeigt.

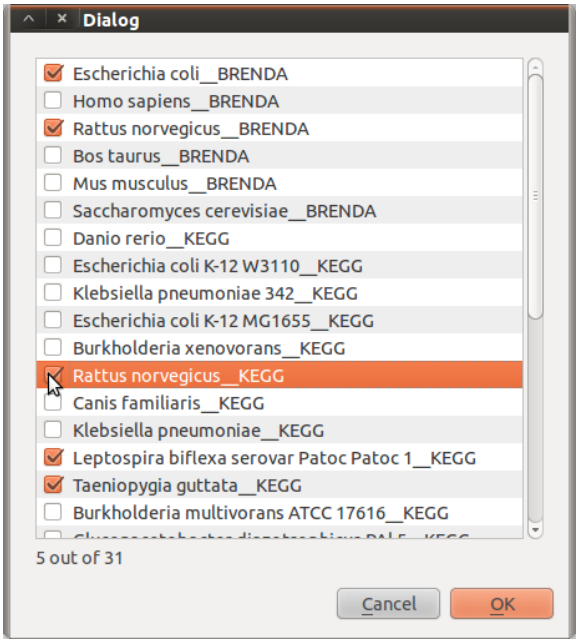

Abbildung 4.49: Durch Verbindung mit der „metabolic_pathways“ Datenbank (siehe Kapitel 3.9.5) werden verschiedene Organismen durch ein Dialogfenster auswählbar. Es können mehrere Organismen gleichzeitig ausgewählt werden.

Menüpunkt „Window“

Über das „Window“ Menü in der Menüleiste lassen sich die erstellten Netzwerk-Fenster der MDI verwalten. Sie können kaskadiert oder gekachelt angeordnet werden. Ebenso kann der Benutzer durch verschiedene Netzwerke navigieren, sie aktualisieren oder schließen.

Menüpunkt „Plugins“

MapOmnia ist erweiterbar angelegt und verfügt über Schnittstellen, für die Plugins implementiert werden können. Diese können unter dem Menüpunkt „Plugins“ zur Laufzeit geladen werden und werden dann automatisch in den Plugin-Ordner des Programms kopiert. Informationen über die eingebundenen Plugin-Funktionen werden grafisch ausgegeben, und ihre Funktionalität lässt sich durch neue erstellte Menüpunkte sowie Toolbars anwählen.

Menüpunkt „Scripts“

Das Menü verfügt über ein „Recent Scripts“-Untermenü. Dieses listet die Skripte, die der Nutzer in „Network Script Fensters“ gespeichert hat. Beim Öffnen von Netzwerk-Dateien werden die erzeugten Netzwerkobjekte aus Performancegründen nicht der Skript-Engine übergeben. Sollen diese durch Skripte manipulierbar werden, so kann dieses durch Klicken auf „Add All to Engine“ bzw. „Add Selected to Engine“ geschehen.

Menüpunkt „Settings“

Der Menüpunkt Settings beinhaltet die Konfigurationsmöglichkeiten des Programms MapOmnia. Fast alle zentralen Einstellungen werden im Konfigurations-Dialogfenster vorgenommen. Dieses Dialogfenster besitzt auf der linken Seite eine Listenansicht. Diese Ansicht dient der Navigation durch die einzelnen Konfigurationsmenüs. Hierbei werden zunächst die Listenpunkte angezeigt, die zum Basisprogramm MapOmnia gehören. Diese sind:

- General
- Biochemistry Data,
- Database Connection
- Selection Properties
- GraphicsScene
- GraphicsView

Weitere Listenelemente sind die dynamisch geladenen Plugins. Im Folgenden werden die Konfigurationsmenüs beschrieben, die bereits zur Basisversion von MapOmnia gehören.

Im obersten Listenpunkt „General“ werden allgemeine Programmeinstellungen vorgenommen. Hierzu gehören die Auswahl der Sprache sowie das generelle Erscheinungsbild (siehe Abbildung 4.50).

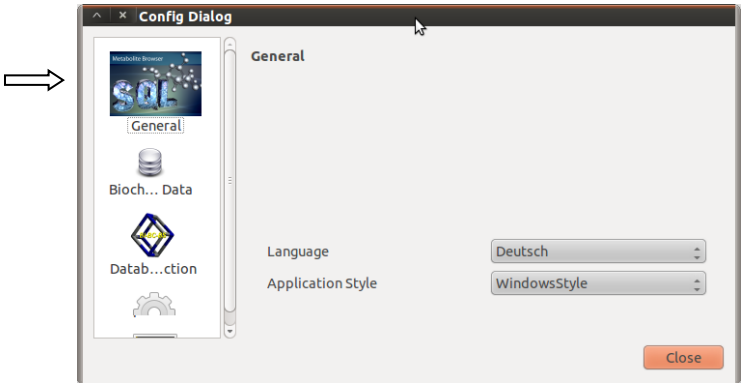

Abbildung 4.50: Dialogfenster Konfiguration: In der Konfigurationsoption „General“ werden allgemeine Programmeinstellungen vorgenommen. Hierunter finden sich die Spracheinstellung sowie die Einstellung für das Erscheinungsbild.

Im zweiten Listenpunkt „Biochemistry Data“ können die Einstellungen für die Datenbank-Verlinkungen verändert werden (vergleiche Kapitel 4.1.4.6). Abbildung 4.51 zeigt das dazugehörige Dialogfenster mit vordefinierten SQL-Abfragen.

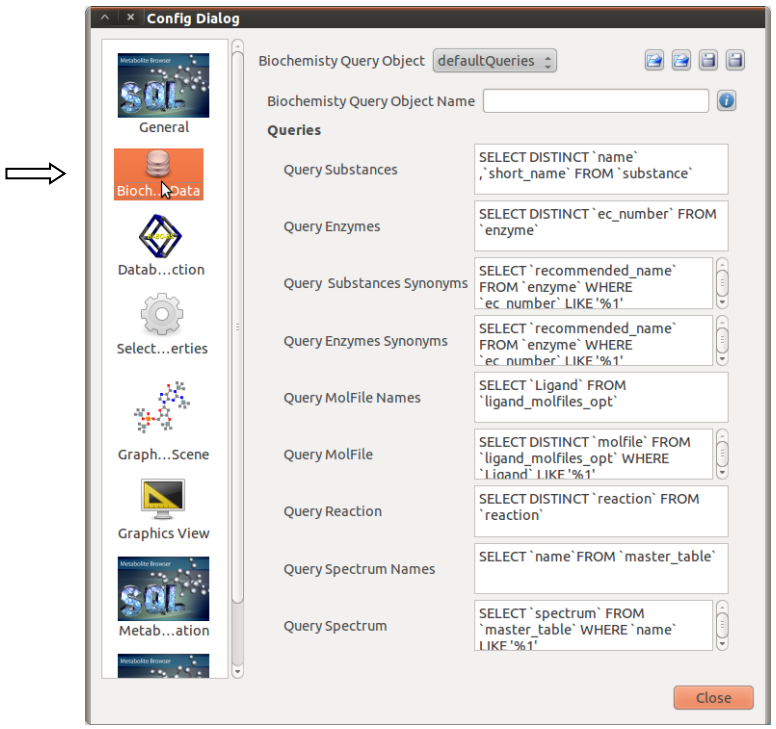

Abbildung 4.51: Dialogfenster Konfiguration: In der Konfigurationsoption „Biochemistry Data“ können SQL Anfragen formuliert werden. Erstellte Einstellungen lassen sich im Ini- und Binär-Format speichern.

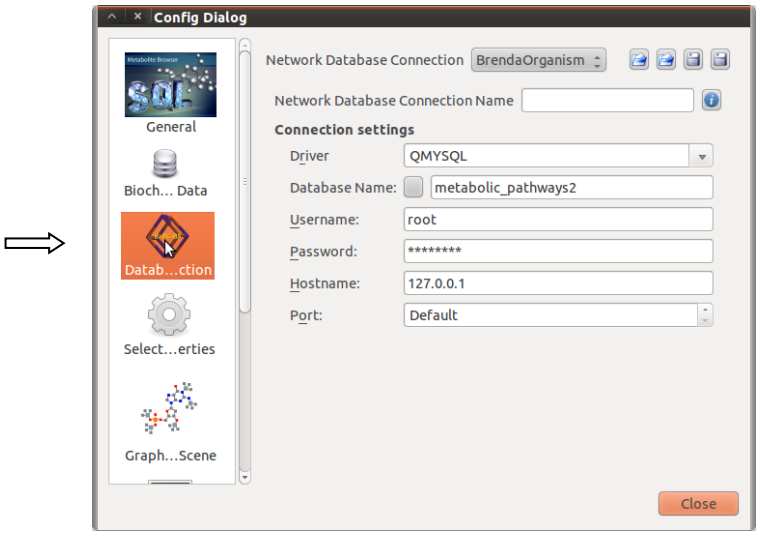

Abbildung 4.52: Dialogfenster Konfiguration: In der Konfigurationsoption „Database Connection“ können Datenbank-Verbindungseinstellungen vorgenommen werden. Erstellte Einstellungen lassen sich im Ini- und Binär-Format speichern.

Der dritte Listenpunkt „Database Connection“ öffnet die Optionen für die Datenbank-Verbindungen. Hier werden der Datenbank-Treiber, der Datenbank-Name, der Host-Name, Port sowie die Benutzerdaten eingetragen. Die Einstellungen lassen sich abspeichern und laden (vergleiche Abbildung 4.52).

Wie bereits in im Kapitel 4.1.5.2 erwähnt, werden im „Network Visualisation Fenster“ selektierte Netzwerkobjekte ebenfalls im „Network Data Table Browser“ angezeigt. Hierbei werden die Attribute und ihre Attributwerte in Tabellen visualisiert. Der Benutzer kann die Attribute vorgeben, die er in der Tabelle dargestellt möchte. Dieses geschieht unter dem Listenpunkt „Selection Properties“ im Konfigurationsdialogfenster. Die hierbei möglichen, einstellbaren Attribute beziehen sich nur auf die statischen, vom Programm vorgesehenen Attribute. Dynamisch geladene oder selbst zugewiesene Attribute werden nicht berücksichtigt. Die Zuweisung erfolgt durch Aktivierung einer Checkbox für das jeweilige Attribut. Dieses ist exemplarisch in Abbildung 4.53 gezeigt.

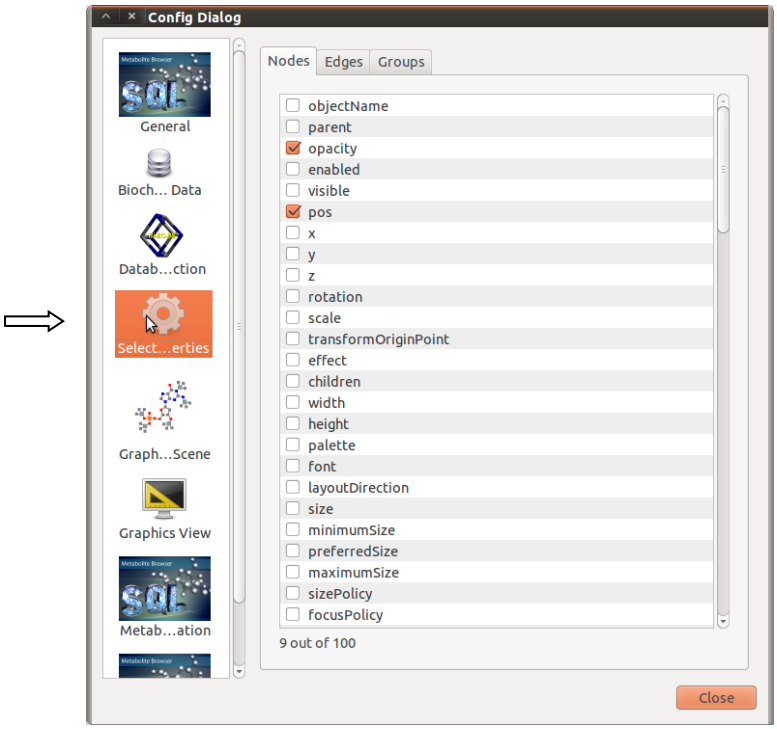

Abbildung 4.53: Dialogfenster Konfiguration: In der Konfigurationsoption „Selection Properties“ kann der Benutzer die Attribute von Netzwerkobjekten vorgeben, welche bei Selektion dieser im „Network Visualisation Fenster“ im „Network Data Table Browser“ angezeigt werden sollen. Die Auswahl erfolgt über Checkboxen.

Im Listenpunkt „GraphicsScene“ können Einstellungen für die „Network Graphics Scene“ vorgenommen werden. Die „Network Graphics Scene“ ist die Umgebung, in der die Netzwerkobjekte positioniert werden. Sie wird im „Network Visualisation Fenster“ angezeigt. Der Benutzer kann das Hintergrundverhalten einstellen, indem er ein Raster definiert, oder einen Hintergrund auswählt. Neben der Möglichkeit ein Brush zu definieren, kann er ebenfalls ein Hintergrundbild laden. Wie bei anderen Einstelloptionen lassen sich auch hier die Einstellungen speichern und laden (vergleiche Abbildung 4.54).

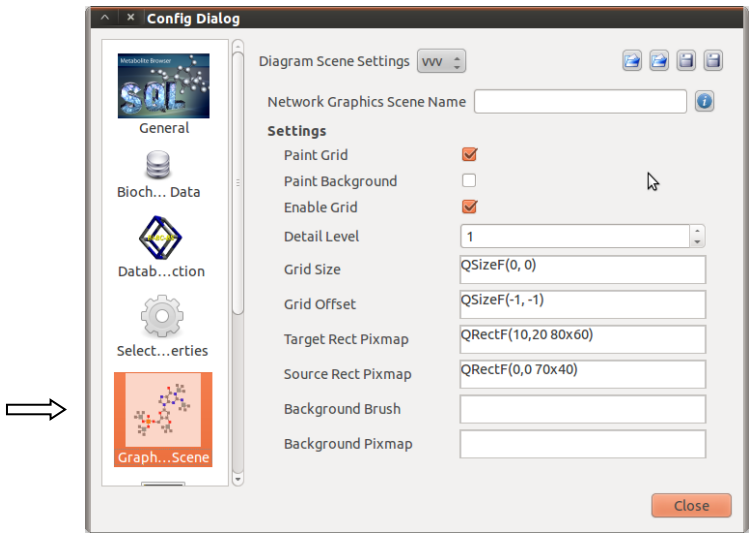

Abbildung 4.54: Dialogfenster Konfiguration: In der Konfigurationsoption „GraphicsScene“ werden sämtliche Einstellungen der „Network Graphics Scene“ vorgenommen. Erstellte Einstellungen lassen sich im Ini- und Binär-Format speichern.

Im letzten zum Programm MapOmnia gehörenden Listenpunkt „GraphicsView“ werden die Einstellungen verändert, die zu dem „Network Visualisation Fensters“ gehören. Hier kann festgelegt werden, ob OpenGL für das Rendering benutzt werden soll und ob die Objekte antialiasiert (Kantenglättung) werden sollen. Das Konfigurationsmenü ist in Abbildung 4.55 dargestellt.

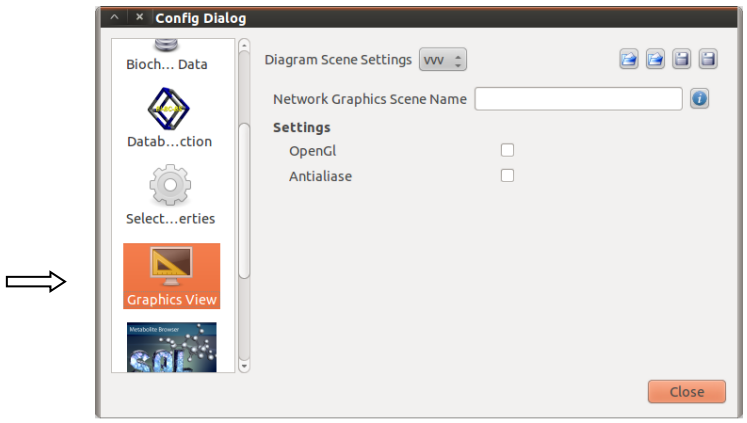

Abbildung 4.55: Dialogfenster Konfiguration: In der Konfigurationsoption „GraphicsView“ werden sämtliche Einstellungen des „Network Visualisation Fensters“ vorgenommen. Einstellungen lassen sich im Ini- und Binär-Format speichern.

Neben den Einstellmöglichkeiten im Konfigurationsdialog finden sich unter dem Menüpunkt „Settings“ noch die Detektionseinstellungen. Die Detektionserkennung kann für alle Netzwerkobjekte aktiviert und deaktiviert werden. Sofern die Detektionserkennung aktiv ist, werden Netzwerkobjekte, die mit anderen Netzwerkobjekten kollidieren, grün hinterlegt dargestellt.

Menüpunkt „Help“

Unter dem Menüpunkt „Help“ können Informationen zu MapOmnia, Qt und zu geladenen Plugins angezeigt und abgerufen werden. Abbildung 4.56 zeigt exemplarisch das Informationsfenster für geladene Plugins.

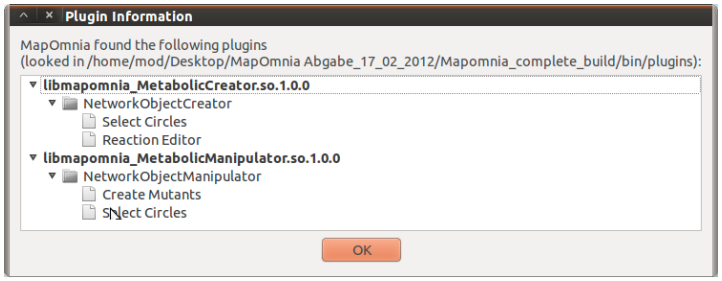

Abbildung 4.56: Informationsfenster für die geladenen Plugins. Es wird hier die jeweilige Schnittstelle, für das geladene Plugin und dessen Methoden angezeigt.

4.1.5.16 Weitere Dialogfenster

Um die Benutzerfreundlichkeit der Applikation MapOmnia zu erhöhen, wurden zahlreiche Dialogfenster, welche diverse Einstellungen GUI-basiert übernehmen, implementiert. Insbesondere durch die Verwendung der im Kapitel 4.1.4.3 beschriebenen Delegate-Programmierung sind Editoren für verschiedene Datentypen implementiert worden. Im Folgenden werden kurz ausgewählte, selbst entwickelte Dialogfenster beschrieben.

Dialogfenster „Pen“

Das Dialogfenster „Pen“ dient als Editierfenster für die Rahmenlinie von Netzwerkobjekten. Es beherbergt sämtliche Konfigurationsmöglichkeiten, um die Rahmenlinie benutzerspezifisch anpassen zu können. Ein kleines Vorschaufenster rendert automatisch die aktuelle Linie. Abbildung 4.57 zeigt das Dialogfenster.

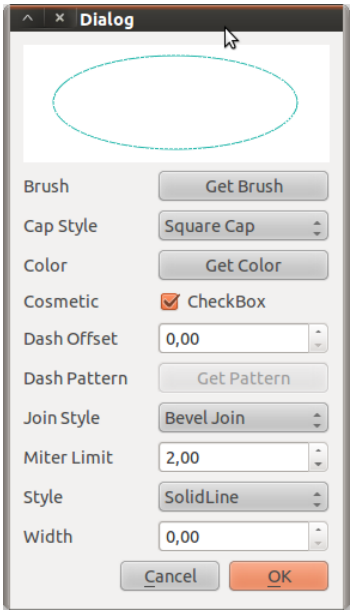

Abbildung 4.57: Das „Pen“-Dialogfenster bietet sämtliche Einstellmöglichkeiten für die Anpassung der Rahmenlinie. Ein Vorschaufenster zeigt die aktuelle Linienart an.

Dialogfenster „Brush Editor“

Möchte der Benutzer das Füllverhalten von Objekten anpassen, kann dieses im Dialogfenster „Brush“ geschehen. Hier werden sämtliche Einstellmöglichkeiten für das

Füllverhalten angeboten. Auch hier dient ein kleines Fenster als Vorschau. Abbildung 4.58 zeigt das Dialogfenster.

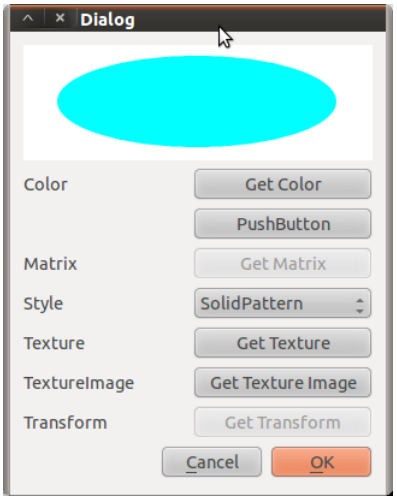

Abbildung 4.58: Das „Brush“ Dialogfenster bietet sämtliche Konfigurationsmöglichkeiten inklusive eines Vorschaufensters für die Anpassung des Füllverhaltens.

Dialogfenster „ListItemSelection Editor“

Sofern in MapOmnia Listen mit unterschiedlichen Datentypen verwaltet oder verändert werden, kann dieses innerhalb des „ListItemSelection Editors“ erfolgen. Die unterschiedlichen Datentypen werden innerhalb des von Qt bereitgestellten Datentyps *QVariant* gekapselt. Für die Erkennung und Repräsentation wurde eigens der „VariantEditor“ implementiert. Dieser erkennt automatisch den vorliegenden Datentyp und stellt diesen entsprechend dar. Dadurch ist es möglich, Listen unterschiedlicher Datentypen zu bearbeiten. In Abbildung 4.59 ist der Editor für die Bearbeitung einer Liste mit Integer-Werten gezeigt. Man kann die Reihenfolge durch „Drag and Drop“ verändern, Listenelemente entfernen und hinzufügen.

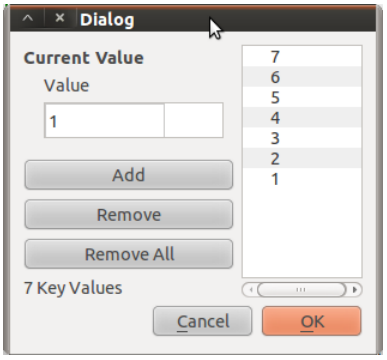

Abbildung 4.59: Innerhalb des „ListItemSelection Editors“ kann die Bearbeitung einer Liste für einen bestimmten Datentyp erfolgen. Im obigen Beispiel wurde eine Liste mit Integer-Werten in den „ListItemSelection Editor“ geladen.

Dialogfenster „Picture Editor“

Der „Picture Editor“ dient der Weiterleitung für die Bearbeitung von Bildern (Abbildung 4.60). Zu diesem Zweck kann entweder direkt das Grafikprogramm Gimp verwendet werden oder der Benutzer wählt ein anderes Programm seiner Wahl aus. Der Editor erkennt, wenn der fremde Prozess beendet wurde, und lädt automatisch das bearbeitete Bild.

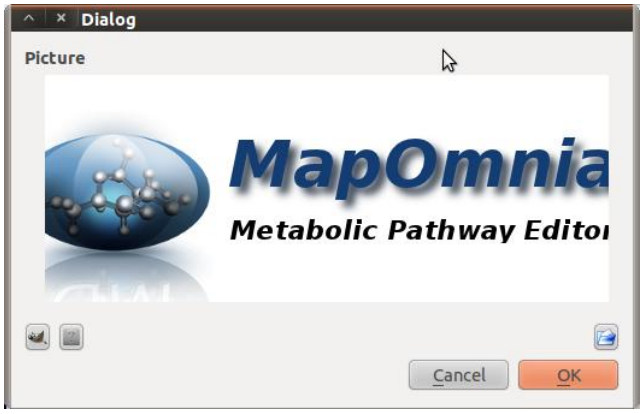

Abbildung 4.60: Der „Picture Editor“ ermöglicht die externe Bild-Bearbeitung in Programmen. Hierbei wird Letzteres innerhalb eines eigenen Prozesses gestartet.

Dialogfenster „Enumeration Editor“

Ein Enumerator (Aufzählungstyp; auch enum) ist ein eigener Datentyp, welcher eine endliche Wertemenge mit zugehörigem konstanten Namen darstellt. Dazu gehören beispielsweise Wochentage oder Farben. Intern werden Enumeratoren als Zahlen codiert. Möchte der Benutzer einen Enumerator-Wert ändern, so muss er die codierte Zahl interpretieren können. Um diese Funktionalität in MapOmnia bereitzustellen, bietet MapOmnia den „Enumeration Editor“ an.

Für alle von *QObject* abgeleiteten Klassen kann durch das *QMetaSystem* ermittelt werden, welche Enumeratoren vorhanden sind und welche Namen den Elementen eines Aufzählungstyps gegeben wurden. Aus diesen Namen wird dynamisch das Dialogfenster aufgebaut und zur Auswahl mit Checkboxen versehen. Abbildung 4.61 zeigt dies exemplarisch für die Veränderung eines Brush Styles.

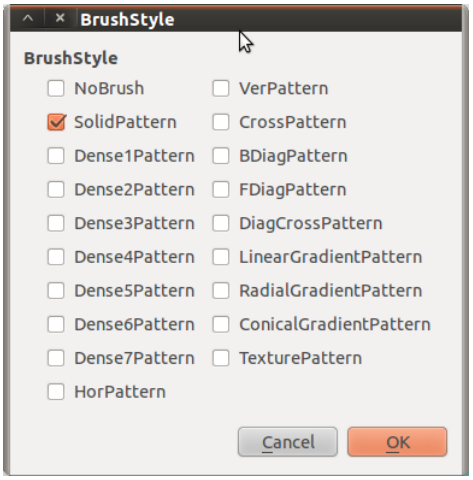

Abbildung 4.61: Der „Enumeration Editor“ wird dynamisch aufgebaut und präsentiert die verwendeten Namen der Elementen eines Aufzählungstyps. Durch Verwendung von Checkboxen sind diese selektierbar.

Neben den beschriebenen Dialogfenstern für die individuelle Bearbeitung von Datentypen wurde ebenfalls ein Dialogfenster „RectangleEditor“ für die Bearbeitung von Rechtecken, das Dialogfenster „PointEditor“ für Veränderungen an zweidimensionalen Punktkoordinaten und das Dialogfenster „SizeEditor“ für eine entsprechende Manipulation von geometrischen Größen mit Breite und Höhe für das Programm implementiert.

4.1.6 Layout von Netzwerken

Unter dem Menüpunkt „Modify“ kann das Dialogfenster für die Durchführung von Layouts (siehe Abbildung 4.63) geöffnet werden. Wie schon in Kapitel 4.1.5.15 erwähnt, werden folgende Layout-Optionen angeboten: „Linear Layout“ (deutsch: Lineare Anordnung), „Grid Layout“ (deutsch: Rasterbasierte Anordnung), „Elliptical Layout“ (deutsch: Elliptische Anordnung), „Painter Path Layout“ (deutsch: Anordnung entlang eines Zeichnungsweges) sowie „Force-Based Layout“ (deutsch: Kräfte- basierte Anordnung).
Eine Übersicht sowie eine kurze Beschreibung findet sich in Tabelle 4.8.
Allen Layout-Funktionen außer dem „Force-Based Layout“ ist gemeinsam, dass sie einen Referenzpunkt haben. Dieser wird zunächst automatisch durch die Berechnung eines geometrischen Schwerpunkts ermittelt. Er kann jederzeit aktualisiert werden, sofern der Benutzer die Schaltfläche (englisch: Button) „Auto“ drückt. Außerdem steht die „Network Graphics Scene“ in ständiger Kommunikation mit der Layout-Funktion, wobei durch Klicken in die Szene automatisch der Referenzpunkt von dieser übertragen wird. Sofern die Parameter der selektierten Layout-Funktion verändert werden, werden für das resultierende Layout, Punkte in der „Network Graphics Scene“ gerendert. Diese sollen als Vorschau für das resultierende Layout-Verhalten dienen. Das Layout selbst wird erst angewandt, nachdem die Schaltfläche „Apply Layout“ betätigt wurde. Hierbei werden die Knoten den Layout-Punkten zugeordnet. Da beide Elementmengen in Listen gespeichert sind, können diese in ihrer Reihenfolge verändert werden. Hierfür befindet sich ein zweites Tab „Item Order“ auf dem Dialog (siehe Abbildung 4.62). Man kann beide Listen randomisieren sowie zirkulieren.

Tabelle 4.8: Übersicht der angebotenen Layout-Funktionen

Layout-Funktion	Beschreibung
Linear Layout	Erzeugt lineare Layout-Punkte für die Ausrichtung von Knoten.
Grid Layout	Erzeugt Raster Punkte für die Ausrichtung von Knoten.
Elliptical Layout	Erzeugt Punkte für die Ausrichtung von Knoten, welche auf einer Ellipse liegen.
Painter Path Layout	Erzeugt Punkte für die Ausrichtung von Knoten, die auf einem Textweg liegen.
Force-Based Layout	Für die Ausrichtung von Knoten wird ein kräftebasierter Algorithmus verwendet.

Im Dialogfenster wird die Möglichkeit angeboten, ein durchgeführtes Layouting zwischenzuspeichern. Der initiale Zustand der Knoten wird bei Aufruf des Layout-Dialogfensters automatisch gesichert.

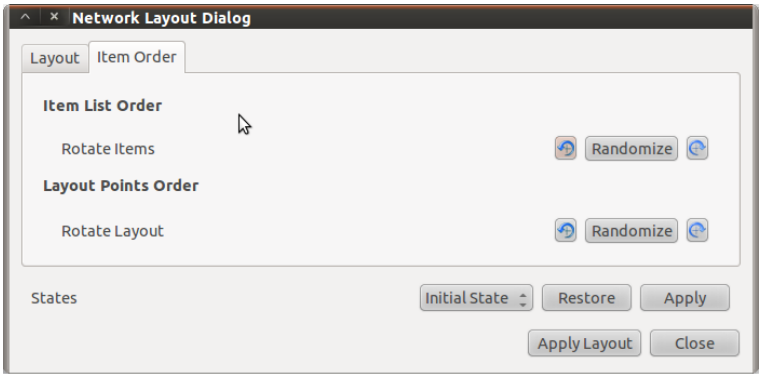

Abbildung 4.62: Im Layout Dialogfenster bietet der rechte Tab die Möglichkeit, die Reihenfolge der Knoten sowie die Layout-Punkte zu verändern.

„Linear Layout“

Wurden Knoten selektiert, so können sie linear in horizontaler und vertikaler Orientierung ausgerichtet werden. Die Schrittweite ist hierbei über die Anzahl der selektierten Knoten mit der Breite gekoppelt. Analog gilt dieses auch für die Weite, die durch die Schrittweite und der Anzahl definiert wird. Abbildung 4.63 zeigt die Konfigurationsmöglichkeiten für das lineare Layout.

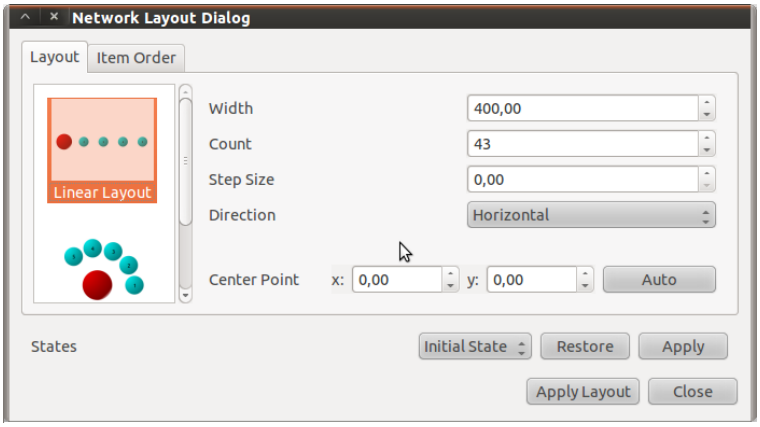

Abbildung 4.63: Das Layout Dialogfenster mit ausgewähltem „Linear Layout“: Hier kann die Weite, die Anzahl sowie die Orientierung angepasst werden.

„Grid Layout“

Möchte der Benutzer die selektierten Knoten an Punkten eines Rasters ausrichten, so erfolgt dieses unter „Grid Layout“. In dem Konfigurationsfenster (siehe Abbildung 4.64) lassen sich die Gesamtbreite und Höhe sowie eine Schrittweite in x- und y- Richtung definieren. Die Anzahl der Rasterpunkte lässt sich ebenfalls einstellen. Die berechneten Rasterpunkte werden in Form einer Liste zurückgegeben, wobei hierbei die Möglichkeit besteht, die Reihenfolge festzulegen. So befinden sich in dem Auswahlfeld (englisch: Combobox) „List Mode“ verschiedene Optionen für die Variation der Reihenfolge der Punkte. Um die Reihenfolge in der „Network Graphics Scene“ zu erkennen, werden an den gerenderten Rasterpunkten Nummerierungen eingefügt. Die Möglichkeiten, die Reihenfolge anzupassen, sind in Tabelle 4.9 aufgeführt.

Auch hier ist die Schrittweite in x Richtung über die Anzahl der selektierten Knoten mit der Breite gekoppelt. Dieses gilt analog für die Schrittweite in y-Richtung. Ein Raster kann beliebige Dimensionen annehmen, wobei es minimal mindestens so viele Punkte haben muss, wie Knoten gewählt wurden.

Tabelle 4.9: Übersicht der Modi für den Aufbau des Rasters

Name der Listen Option	Aufbau des Rasters
LeftRight	Reihenweise von links nach rechts
RightLeft	Reihenweise von rechts nach links
LeftRightRightLeft	Ungerade Reihen: links nach rechts; gerade Reihen: rechts nach links
RightLeftLeftRight	Ungerade Reihen: rechts nach links; gerade Reihen: links nach rechts
UpDown	Spaltenweise von oben nach unten
DownUp	Spaltenweise von unten nach oben
UpDownDownUp	Ungerade Spalten: oben nach unten; gerade Spalten: unten nach oben
DownUpUpDown	Ungerade Spalten: unten nach oben; gerade Spalten: oben nach unten
Random	Zufällig Reihenfolge

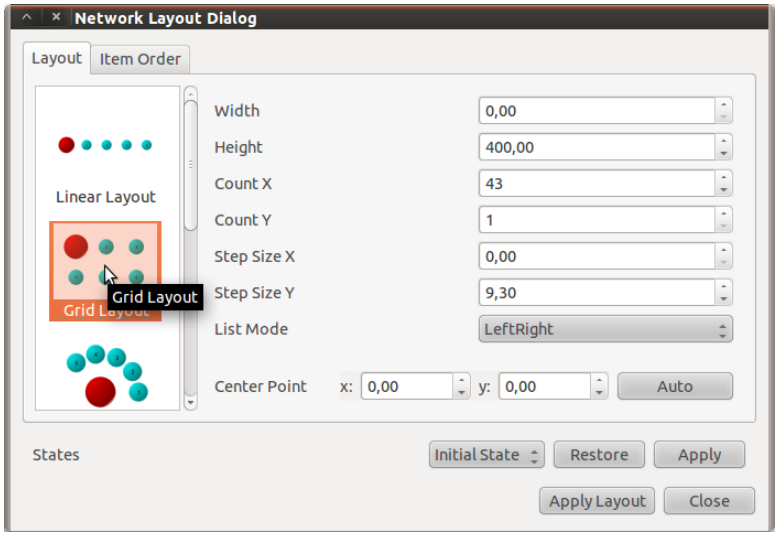

Abbildung 4.64: Das Layout-Dialogfenster mit ausgewähltem „Grid Layout“: Der Benutzer kann sämtliche Parameter des „Grid Layout“ ändern.

„Elliptical Layout“

Knoten können ebenfalls kreisförmig oder elliptisch angeordnet werden. Hierzu wählt man „Elliptical Layout“ im Layout-Dialogfenster. In dem Einstellungsfenster (siehe Abbildung 4.65) können die Breite sowie die Höhe der Ellipse bestimmt werden. Auch lässt sich der Startwinkel, der Endwinkel sowie eine Schrittweite definieren.

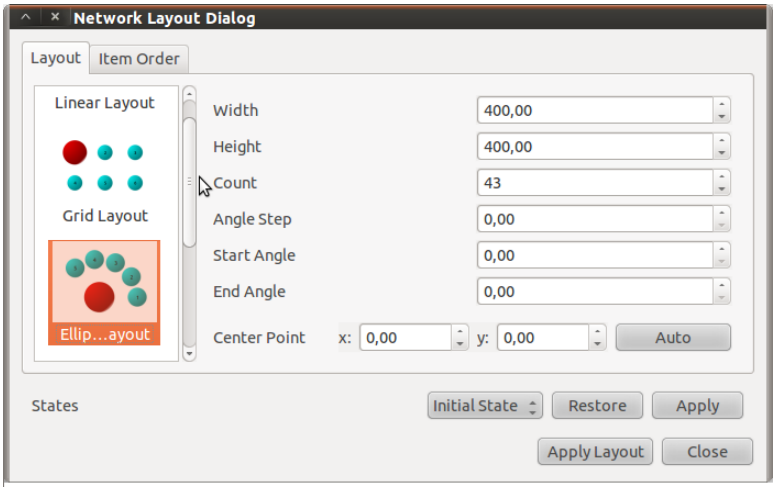

Abbildung 4.65: Das Layout Dialogfenster mit ausgewähltem „Elliptical Layout“: Der Benutzer kann sämtliche Parameter des „Elliptical Layout“ ändern.

„Painter Path Layout“

Wählt der Benutzer das „Painter Path Layout“, kann er die selektierten Knoten entlang von Schrifttexten ausrichten lassen. Hierfür stehen ihm verschiedene Schriftfonts sowie alle verfügbaren Zeichen zu Verfügung. Durch Veränderung der Schrittweite werden die Layout-Punkte so lange entlang des Textes verschoben, bis sie sich über den gesamten Textweg erstrecken. Die Breite des Textes wird durch die Weite einstellbar. In Abbildung 4.66 das „Painter Path Layout“-Menü dargestellt.

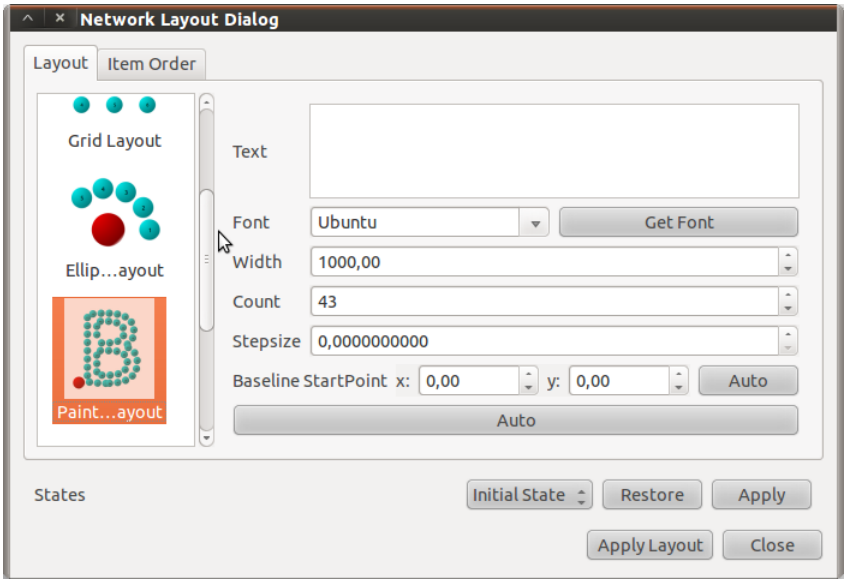

Abbildung 4.66: Das Layout Dialogfenster mit ausgewähltem „Painter Path Layout“: Der Benutzer kann alle Parameter des „Painter Path Layout“ ändern.

„Force-Based Layout“

Kräftebasiertes Ausrichten wird unter dem Auswahlpunkt „Force-Based Layout“ angeboten. Im Gegensatz zu den anderen Layout-Funktionen wird hier kein Referenzpunkt übergeben. Für die iterative Berechnung kann der Benutzer den Wert „Spring“ (deutsch: die Feder), „Charge“ (deutsch: die Ladung), „Damping“ (deutsch: die Dämpfung) sowie die Iterationsanzahl vorgeben. Die Berechnung erfolgt, sobald die Schaltfläche „Run“ gedrückt wird. Die resultierende Ausrichtung kann hierbei animiert oder nicht animiert angezeigt werden. Abbildung 4.67 zeigt die Konfigurationsmöglichkeiten.

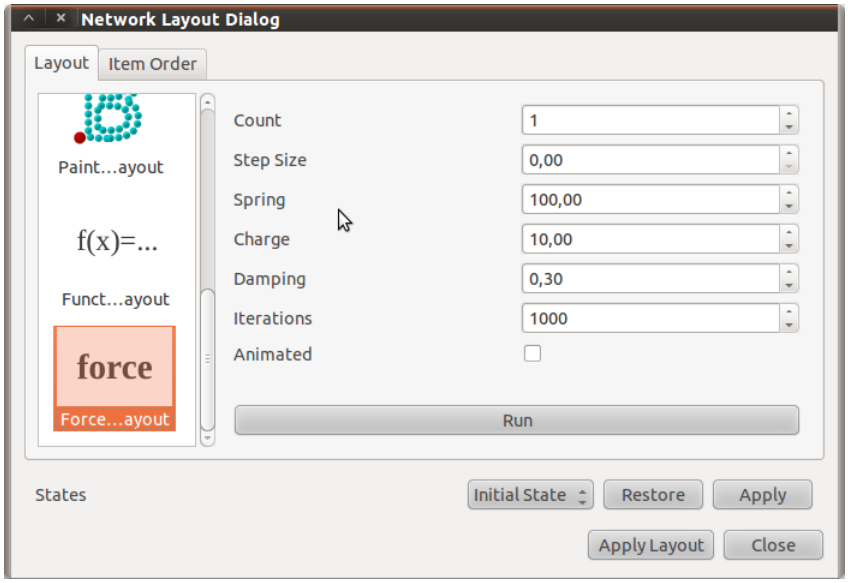

Abbildung 4.67: Das Layout Dialogfenster mit ausgewähltem „Force-Based Layout“: Der Benutzer kann sämtliche Parameter des „Force-Based Layout“ ändern.

4.1.7 Anwendungsbeispiele von MapOmnia

Einige Anwendungsmöglichkeiten von MapOmnia wurden bereits in den vorhergehenden Kapiteln erörtert und zusätzlich mit Beispielen unterlegt. Im Folgenden werden vier interessante Anwendungsfälle vorgestellt, die weitere Programmfunktionen im Detail aufzeigen. Zunächst wird ein synthetischer Organismus erstellt, welcher Eigenschaften mit anderen Organismen teilt (Kapitel 4.1.7.1). Im Anschluss wird in Kapitel 4.1.7.2 die Verwendung der angebotenen Online-Ressourcen aufgezeigt. Kapitel 4.1.7.3 zeigt detailliert, wie experimentelle Daten visualisiert werden können. Abschließend wird in Kapitel 4.1.7.4 die Verwendung der Skript-Funktionalität des Programmes beschrieben.

4.1.7.1 Konstruktion eines neuen Teilnetzwerkes

MapOmnia verfügt über verschiedene Möglichkeiten, ein neues Teilnetzwerk zu erstellen. Dieses kann unter anderem bei bestehender Selektion von Netzwerkobjekten innerhalb des „Network Visualisation Fenster“ einfach über den Menüpunkt „Create new Mapping“ erfolgen.

Eine weitere Möglichkeit, ein neues Teilnetzwerk zu erstellen, bietet der in Kapitel 4.1.5.13 beschriebene „Network Object Selector“. Dieses Dialogfenster ermöglicht Modellierern die Kombination vorhandener Netzwerke. Es öffnet sich das in Abbildung 4.27 dargestellte Fenster, und der Benutzer kann durch sämtliche geöffneten Teilnetzwerke, durch die Gesamtkarte und durch die Gruppen navigieren. Über Auswahlfelder wählt der Benutzer diejenigen Netzwerkobjekte, die in dem neuen, zu erstellenden Netzwerk enthalten sein sollen. Abbildung 4.68 zeigt exemplarisch für die hier aufgezeigte Konstruktion des *in silico*-Organismus, die Auswahl der Glykolyse und Glukoneogenese des Organismus *Corynebacterium glutamicum ATCC 13032* der KEGG-Annotation. Wählt der Benutzer einen Baumknoten aus, der selbst Netzwerkobjekte als Kindknoten besitzt, so werden diese ebenfalls in das neue Netzwerk übernommen. In Tabelle 4.10 ist gezeigt, welche Komponenten für die Erstellung des *in silico*-Organismus ausgewählt wurden und in Abbildung 4.69 ist das erstellte Teilnetzwerk dargestellt.

Tabelle 4.10: Übersicht der Komponenten für die Erstellung des *in silico*-Organismus

Ursprungs-Netzwerk	**Stoffwechselweg**	**Taxonomische Gruppe**
BRENDA-Maps	CO_2-Fixierung in Crenarchaeota	Archaea
Corynebacterium glutamicum ATCC 13032	Glykolyse	
Corynebacterium glutamicum ATCC 13032	Glukoneogenese	
Escherichia coli (BRENDA)	Citratzyklus	
Escherichia coli (BRENDA)	Leucin-Metabolismus	
Escherichia coli (BRENDA)	Gluthathion-Metabolismus	
Homo sapiens (BRENDA)	Cholesterol-Biosynthese	Mammalia

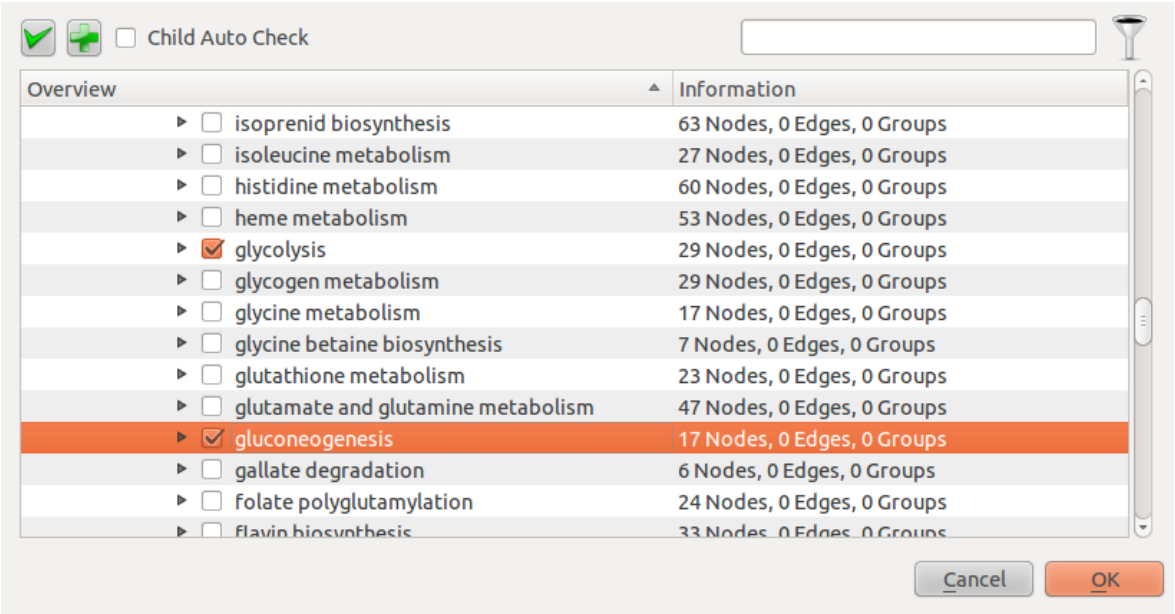

Abbildung 4.68: Im „Network Object Selector“ kann der Benutzer zwischen Teilnetzwerken und Netzwerkobjekten navigieren, und diese einem neuen Teilnetzwerk hinzufügen. Die Erstellung des Netzwerkes wird durch den „OK“-Button initiiert.

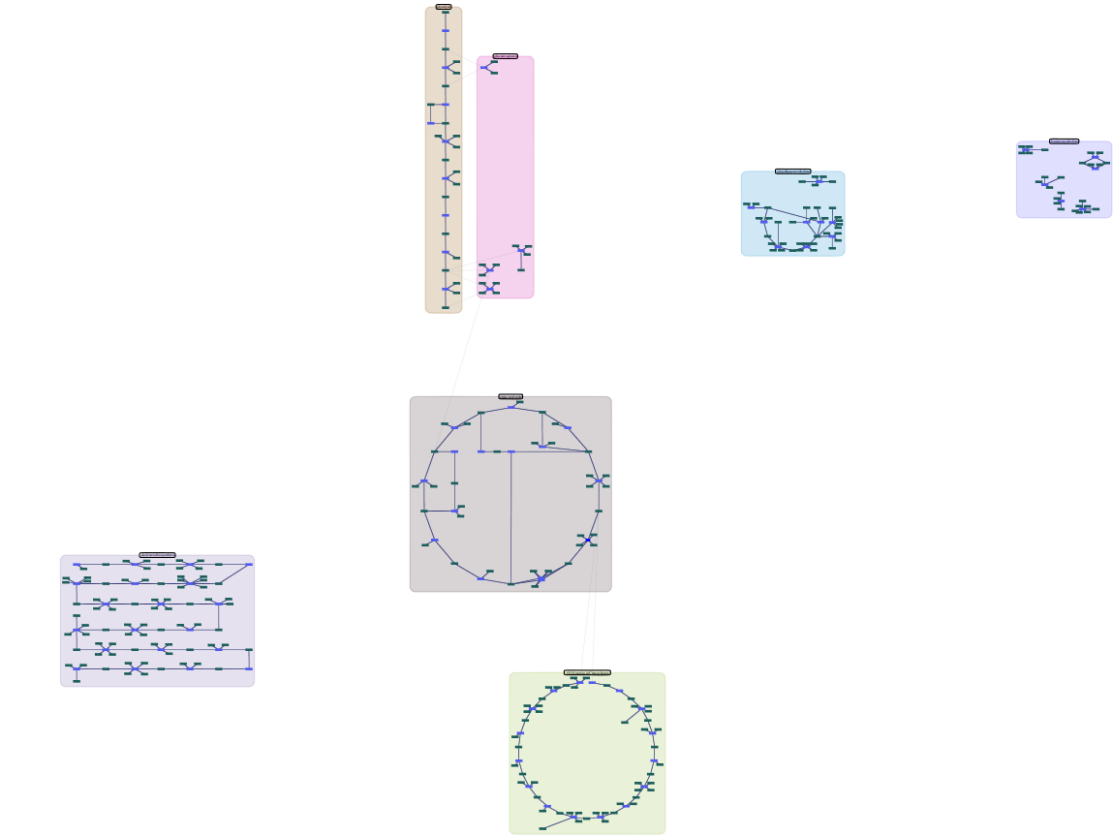

Abbildung 4.69: Das obige Teilnetzwerk ist das Resultat des verwendeten „Network Object Selector“. Es wurden hierbei die CO_2-Fixierung in Crenarchaeota aus der BRENDA-Maps, die Glykolyse, die Glukoneogenese aus C*orynebacterium glutamicum ATCC 13032*, der Citratzyklus, der Leucin- und der Gluthathion-Metabolismus aus *Escherichia coli* (BRENDA-Annotation) und die Cholesterol-Biosynthese aus *Homo sapiens* (BRENDA-Annotation) gewählt.

4.1.7.2 Online Ressourcen verwenden

MapOmnia bietet die Möglichkeit, die BRENDA-Gesamtstoffwechselkarte über eine Datenbankschnittstelle direkt aus BRENDA zu laden (siehe Kapitel 4.1.5.15 unter „Menüpunkt Online Resources“). Neben diesem metabolischen Netzwerk können auch Organismen-Annotationen aus BRENDA oder von PATRIC über eine selbst implementierte Schnittstelle geladen werden. Bei dem Zufügen dieser Teilnetzwerke ist es unerheblich, ob die ursprüngliche Vergleichskarte aus BRENDA stammt, oder der

Benutzer eine eigens erstellte Stoffwechselkarte verwendet. Die Annotation wird hierbei zu dem bestehenden Netzwerk geladen, über Vergleich des angezeigten Knotennamens abgeglichen und dem zu erstellenden Teilnetzwerk hinzugefügt. Im Anschluss werden sämtliche Kanten ergänzt, die in dem Teilnetzwerk im Vergleich zum Referenznetzwerk fehlen und dessen Start- sowie Endknoten vorhanden sind.

In Abbildung 4.70 ist das Dialogfenster gezeigt, welches sich öffnet, wenn der Benutzer die BRENDA-Maps laden möchte. Dem Benutzer werden die vorhandenen Annotationen zur Selektion angeboten. Nach Bestätigen wird zunächst die Gesamtkarte (BRENDA-Maps) geladen, anschließend werden die Teilnetzwerke konstruiert.

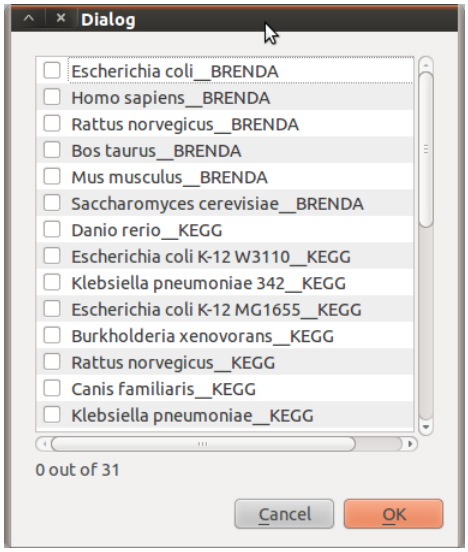

Abbildung 4.70: Möchte der Benutzer die BRENDA-Maps verwenden, so geschieht die entsprechende Auswahl dialogbasiert. Neben der Gesamtkarte können auch organismusspezifische Teilnetzwerke geladen werden.

Falls während der Bearbeitung oder der Betrachtung von Netzwerken auffällt, dass ein organismusspezifisches Netzwerk aus der BRENDA Datenbank fehlt, kann dieses jederzeit nachgeladen werden. Hierbei öffnet sich ein Dialogfenster analog zu dem aus Abbildung 4.70.

Wie schon erwähnt, bietet das Programm außerdem die Möglichkeit, Annotationen aus PATRIC zu laden. Es öffnet sich in diesem Fall das in Abbildung 4.71 dargestellte

Dialogfenster, und der Benutzer kann selektieren, welche Organismen er mit der Gesamtkarte abgleichen und als Teilnetzwerk anzeigen lassen möchte.

Dieses ist im Folgenden exemplarisch für die Organismen *Acinetobacter baumannii 6013113*, *Acinetobacter baumannii UMB003* und *Acinetobacter radioresistens SK82* dargestellt. Von Abbildung 4.72 bis Abbildung 4.74 wird der Abbau der Hexosen für diese drei Organismen sowie ein Ausschnitt der Gesamtstoffwechselkarte (Abbildung 4.75) wiedergegeben.

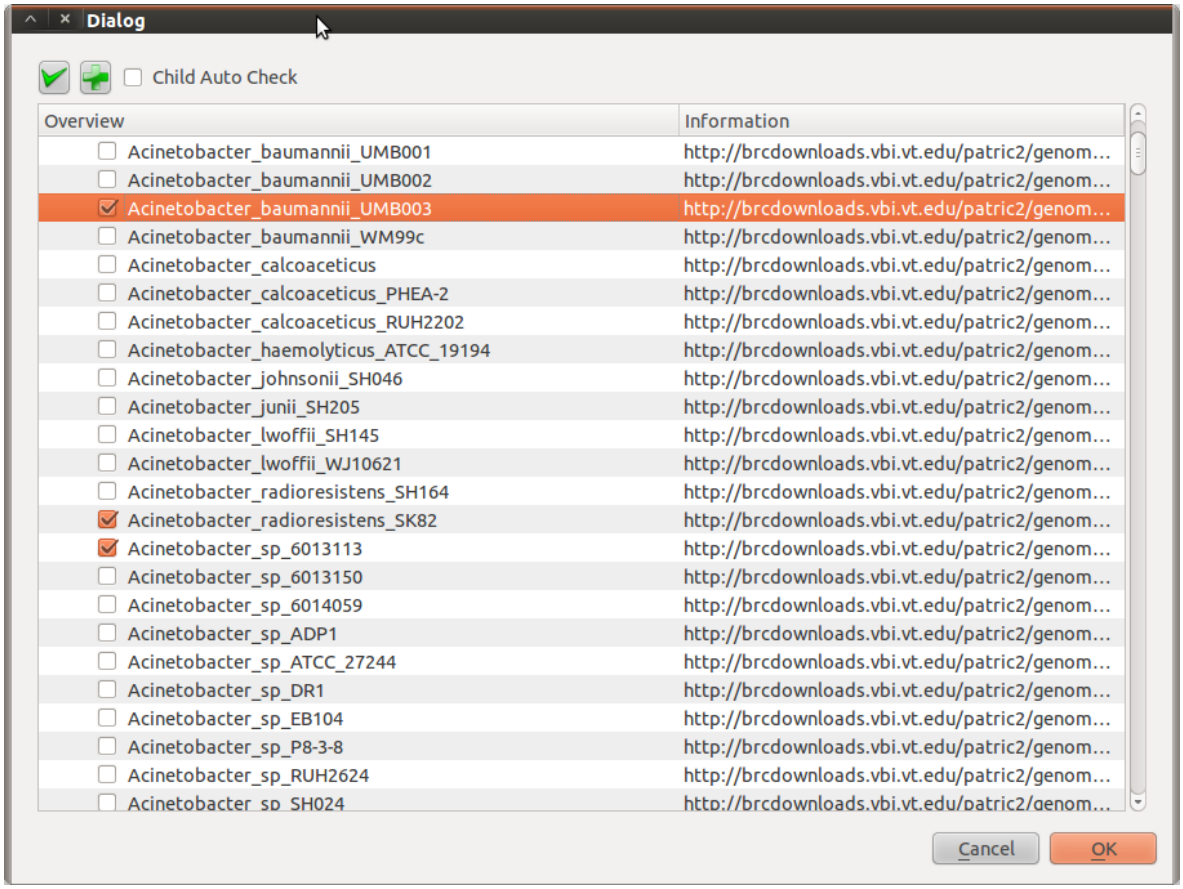

Abbildung 4.71: Der Benutzer kann die gewünschten Organismen selektieren. Die Erstellung der Netzwerke wird durch den „OK“-Button inititiert.

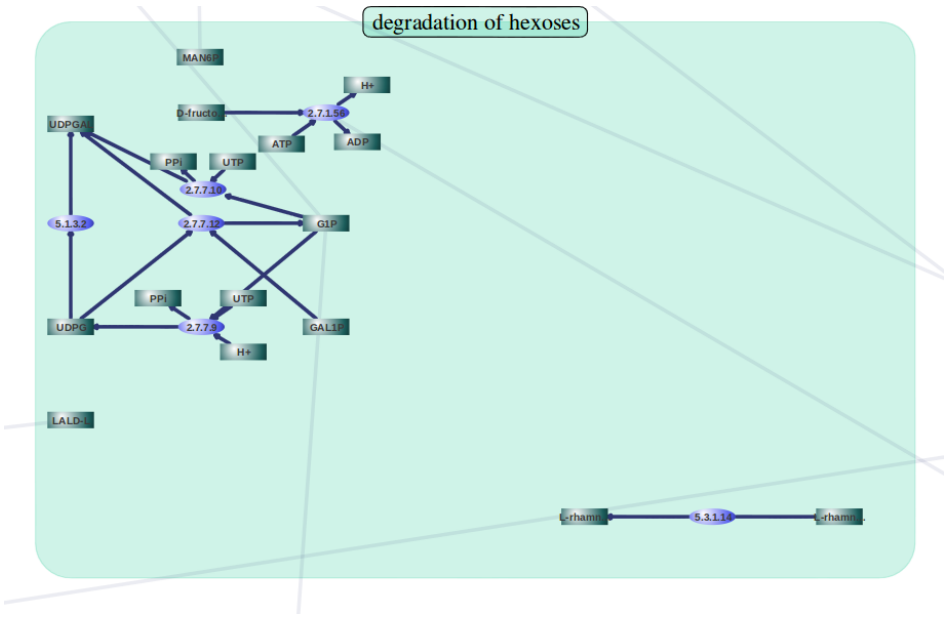

Abbildung 4.72: Die Ansicht zeigt das organismusspezifische Teilnetzwerk von *Acinetobacter baumannii 6013113* geladen aus PATRIC und abgeglichen mit BRENDA-Maps. Es ist der Stoffwechselweg für den Abbau der Hexosen dargestellt.

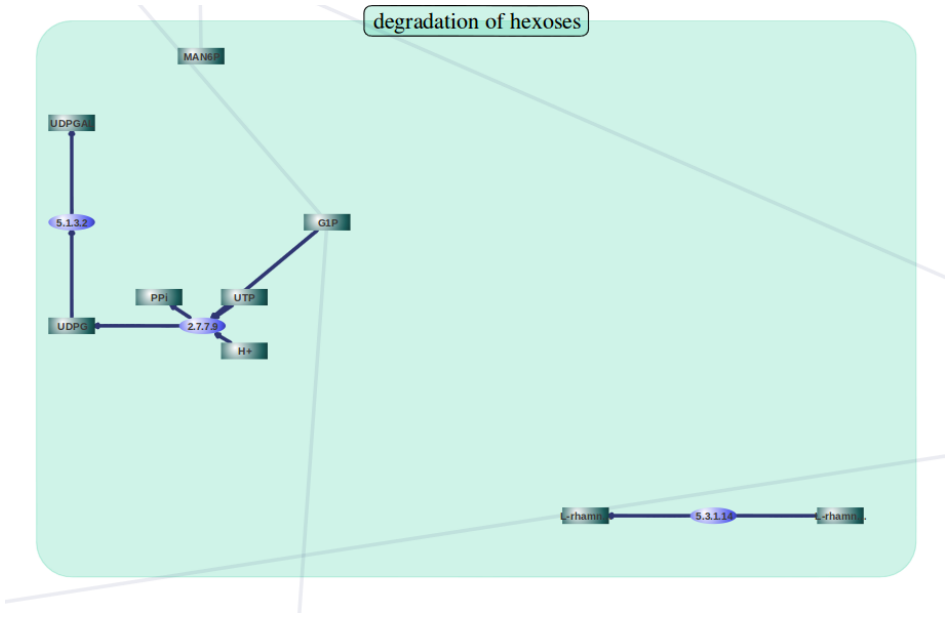

Abbildung 4.73: Die Karte zeigt das organismusspezifische Teilnetzwerk von *Acinetobacter baumannii UMB003*, geladen aus PATRIC und abgeglichen mit BRENDA-Maps. Es ist der Stoffwechselweg für den Abbau der Hexosen dargestellt.

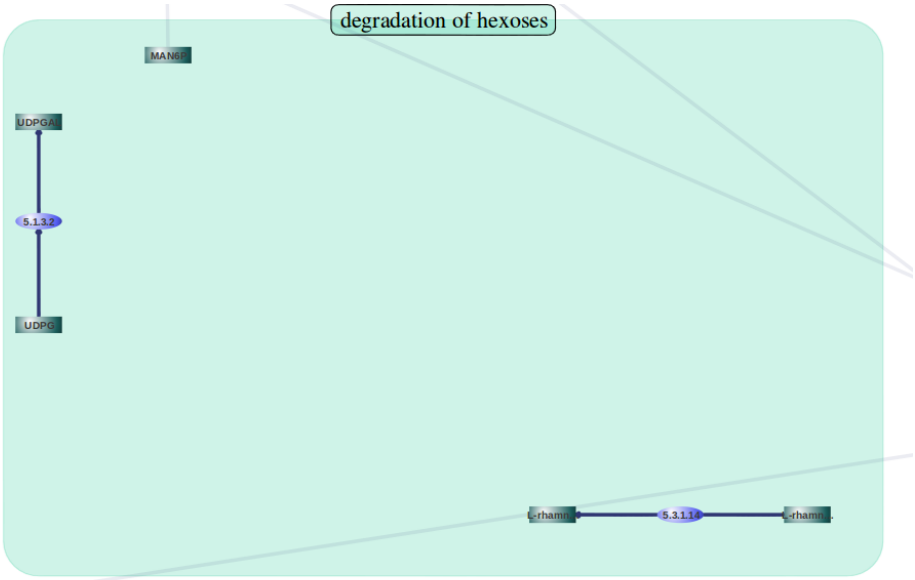

Abbildung 4.74: Die Karte zeigt das organismusspezifische Teilnetzwerk von *Acinetobacter radioresistens SK82*, geladen aus PATRIC und abgeglichen mit BRENDA-Maps. Es ist der Stoffwechselweg für den Abbau der Hexosen dargestellt.

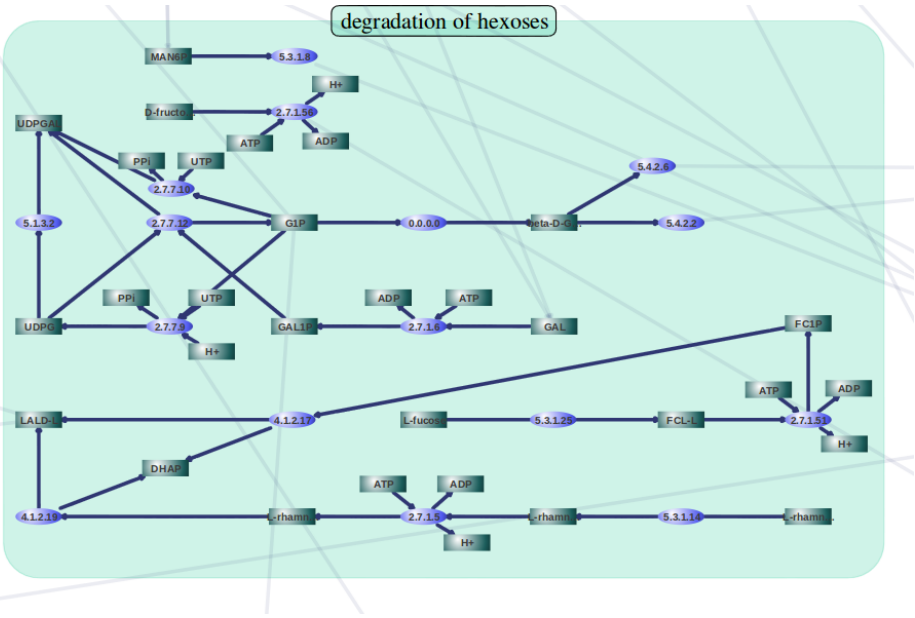

Abbildung 4.75: Die Karte zeigt die BRENDA-Maps. Es ist der Stoffwechselweg für den Abbau der Hexosen dargestellt.

4.1.7.3 Experimentdaten visualisieren

Ein zentrales Ziel von MapOmnia stellt die Möglichkeit dar, experimentelle Daten mit einem Netzwerk abzugleichen und auf dieses abbilden zu können. Um diese Daten in der gewünschten Form zu visualisieren, werden Wizards (siehe Kapitel 4.1.5.14) angeboten, welche den Benutzer schrittweise durch die netzwerkbasierte Abbildung dieser Daten führt. Für die Visualisierung stehen dem Benutzer folgende Methoden zur Verfügung: Er kann die Daten entweder auf die Größe oder auf die Farbe der Knoten abbilden. Sofern es sich um Fluss-basierte Daten handelt, wird die Strichstärke der die Kanten zur Darstellung der Flüsse verwendet.

Im folgenden Beispiel wird für jede dieser Methoden jeweils eine Datenabbildung durchgeführt und das Ergebnis dargestellt. Die Datenquelle „Daten 1“ im Anhang zeigt die verwendete Datengrundlage für die erstellte Projektion der Daten auf die Knoten. Die Flussdaten für die Visualisierung der Flüsse listet die Datenquelle „Daten 2“ im Anhang.

Abbildung 4.76 zeigt den Stoffwechselweg der Sulfat-Reduktion aus BRENDA-Maps im Ausgangszustand.

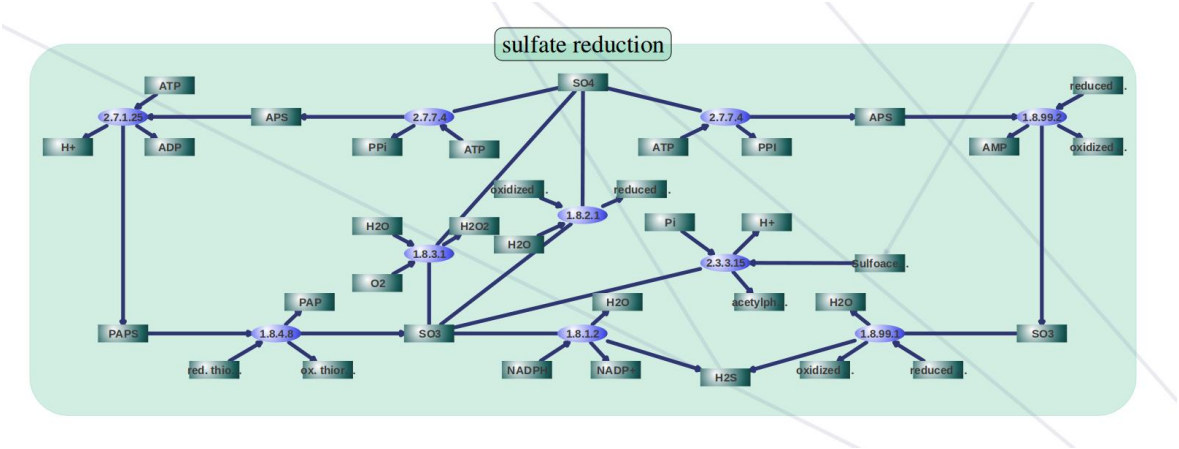

Abbildung 4.76: Die Sulfat-Reduktion aus der generischen BRENDA-Maps Karte.

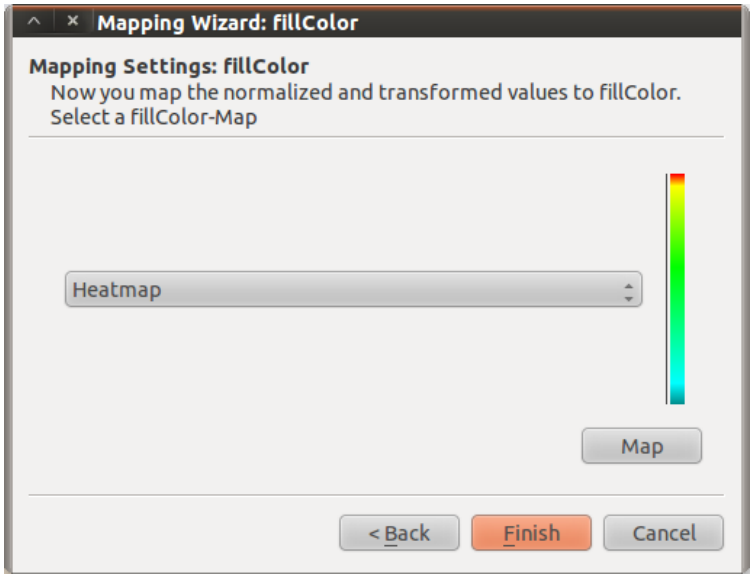

Abbildung 4.77: In dem Fenster ist die Heatmap gewählt, wobei dem Benutzer die erstellten Colormaps als Auswahlliste zur Verfügung stehen. Es werden alle Colormaps zur Verfügung gestellt, die der Benutzer gespeichert hat.

Es wurde zunächst eine Farbcodierung festgelegt, wobei die in Abbildung 4.77 gezeigte Heatmap als Standard-Codierung verwendet wurde. Die Daten wurden hierbei nicht transformiert (vergleiche Kapitel 4.1.5.14). In Abbildung 4.78 ist die resultierende Farbcodierung gezeigt: Die gefundenen Knoten werden ihrem Wert entsprechend eingefärbt. Bei jeder Farbzuweisung werden zwei Teilnetzwerke konstruiert. In das erste wird zusätzlich das gesamte Ausgangsnetzwerk mit aufgenommen, in dem zweiten sind nur die gefundenen Knoten des Teilnetzwerks enthalten.

Im Anschluss an die Farbzuweisung wird auf Basis des neu erstellten Teilnetzwerks mit den gesamten Netzwerkobjekten (Abbildung 4.78 oben) eine Größencodierung durchgeführt. Hierbei ist die Datenbasis dieselbe wie bei der ersten Codierung. In Abbildung 4.79 werden die Rechteckgrößen für die Knotentransformation aufgezeigt. Somit liegt die minimale Rechteckgröße bei einer Breite und Höhe von 10. Die maximale Rechteckgröße bei einer Breite und Höhe von 50. Die transformierten Rechtecke der Knoten, angepasst durch die zugeordneten Werte aus dem Datensatz, sind somit

quadratisch. Abbildung 4.80 zeigt das Ergebnis der oben genannten Codierung. Die Knoten werden gemäß ihrer Werte in der Größe angepasst.

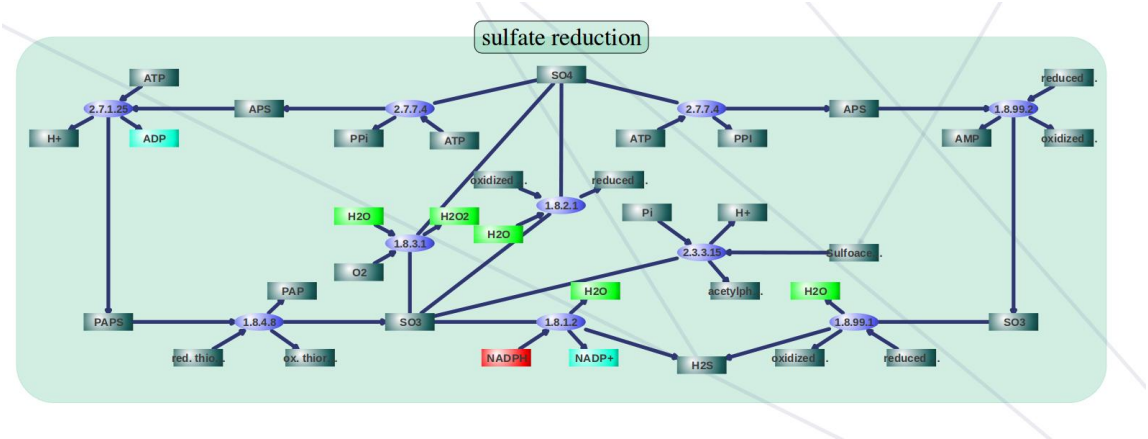

Abbildung 4.78: Bei der Farbzuweisung gemäß ausgewähltem Codierungsschema werden jeweils zwei Teilnetzwerke erstellt. Ein Teilnetzwerk enthält hierbei sämtliche Netzwerkobjekte aus dem Ursprungsnetz (hier dargestellt), auf dem die Datenabbildung ausgeführt wurde. Das zweite Teilnetzwerk enthält ausschließlich die gefundenen Netzwerkobjekte (nicht dargestellt).

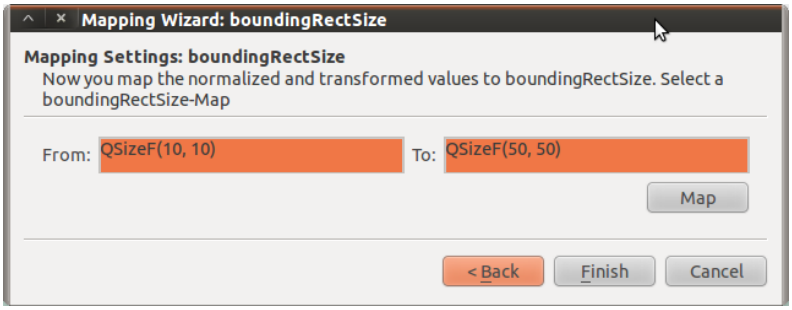

Abbildung 4.79: Bei der Abbildung von Daten müssen jeweils Unter- sowie Obergrenzen für die Transformation angegeben werden. Diese Grenzen sind datentypabhängig. Im oberen Beispiel wird eine Größencodierung durchgeführt, und dementsprechend werden als Grenzen, die minimale und maximale Höhe sowie Breite der Rechtecke benötigt.

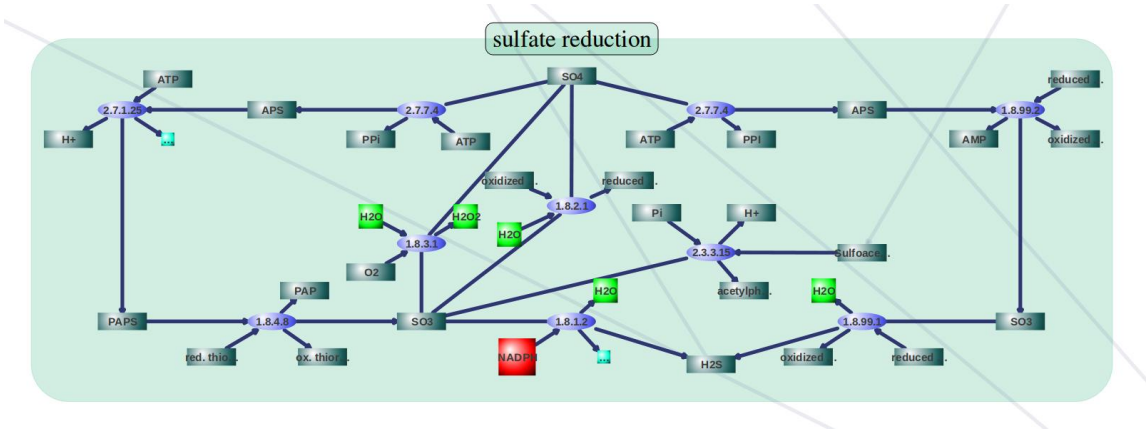

Abbildung 4.80: Auf die generischen BRENDA-Maps Karte wurden experimentelle Daten abgebildet. Hierbei wurden sowohl eine Größen- als auch eine Farbtransformation durchgeführt. Exemplarisch ist der Ausschnitt aus der Karte mit der Sulfat-Reduktion dargestellt.

Abschließend wird eine Fluss-basierte Abbildung mit der Datengrundlage aus „Daten 5“ (im Anhang) durchgeführt. Als Ausgangsnetzwerk dient das zuvor erstellte Netzwerk, bei dem sowohl eine Farb- als auch eine Größentransformation durchgeführt wurden. Die minimale Strichstärke der Kanten wurde auf 5 und die maximale auf 30 gesetzt. Das Ergebnis der Fluss-basierten Datenabbildung ist in Abbildung 4.81 gezeigt. Es wurden hierbei Flüsse enzymatischer Reaktionen im Chlorophyll-Metabolismus angepasst.

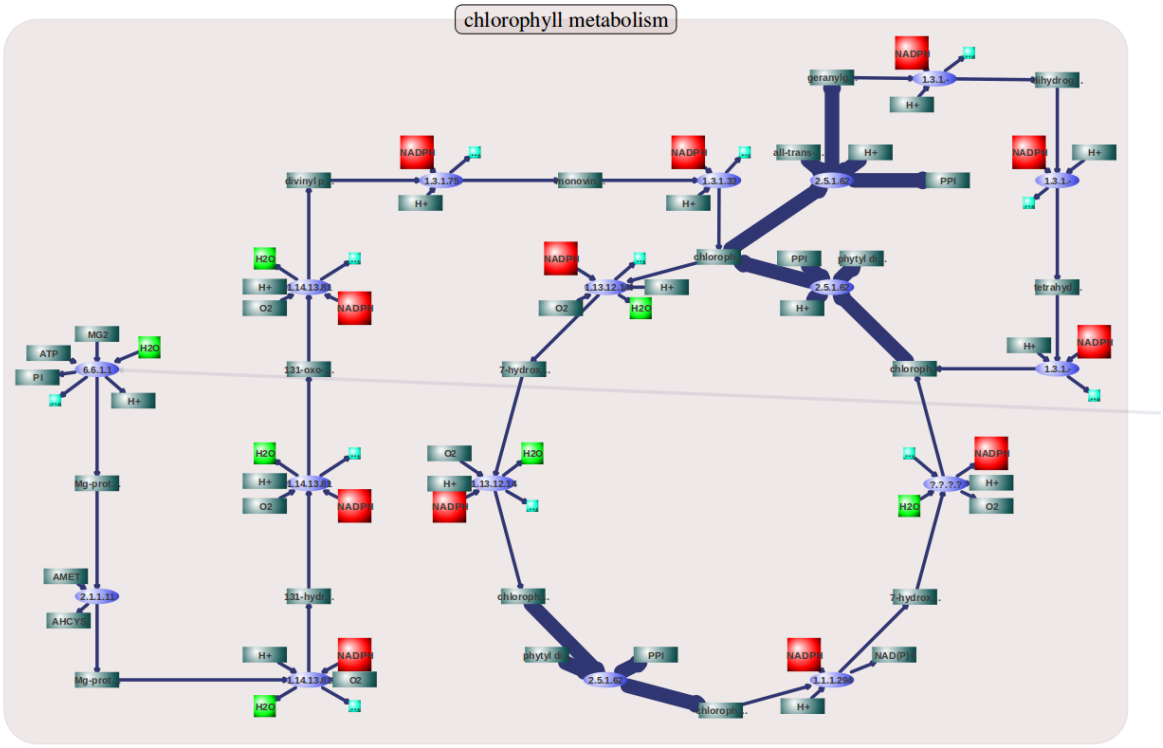

Abbildung 4.81: Zusätzlich zu einer Größen- sowie eine Farbcodierung eines Datensets auf die generischen BRENDA-Maps Karte wurden Flussdaten abgebildet. Es ist der Chlorophyll-Metabolismus dargestellt.

4.1.7.4 Verwendung von Skripten

MapOmnia verfügt über die Möglichkeit, das Programm über Skripte zu steuern und Einfluss auf die Netzwerkobjekte zu nehmen (siehe Kapitel 3.7.2.4 sowie Kapitel 4.1.5.12). Es wurden hierbei die meisten der implementierten Methoden der Netzwerkobjekte der Skript-Engine zugänglich gemacht (siehe Klassenbeschreibungen im Anhang). Im Folgenden ist ein Quellcode-Beispiel aufgeführt (siehe Quellcode 4.2). In diesem Beispiel wurde über die Netzwerkknoten iteriert und überprüft, ob die Knoten einem Enzym entsprechen. Sofern ein Knoten ein Enzym darstellt, wird die Farbe des Knoten auf Rot gesetzt. Sollte es sich um einen Nicht-Enzym-Knoten handeln, liegt, da es sich um ein bipartites Netzwerk handelt, ein Substanzknoten vor. Im weiteren Verlauf wird überprüft, ob der Knoten eine Beschriftung (englisch: Label) besitzt, welche „CO2“ ist. Handelt es sich folglich um Kohlendioxid, wird die Knotenfarbe auf Grün gesetzt und der

Knoten farblich unterlegt (englisch: Highlighted). Ist dies nicht der Fall, wird die Farbe des betreffenden Knoten auf Blau gesetzt. Die Anwendung des beschriebenen Skripts auf ein bestehendes Netzwerk ist in Abbildung 4.82 dargestellt. Es handelt sich zu Illustrationszwecken um ein triviales Beispiel. Durch die Kombination der Eigenschaften und Methoden, die für die Objekte zugänglich gemacht wurden, bieten sich dem Benutzer vielfältige Möglichkeiten Netzwerke zu verändern, zu analysieren und anzupassen.

Quellcode 4.2: Der folgende Quellcode ist ein Skript für die Demonstration der Skript-Funktonalität in MapOmnia. Es differenziert Knoten nach Substanzen und Enzymen, und verändert ihre Farbe. Ebenfalls wird die Substanz Kohlendioxid farblich hervorgehoben

```
var getProperty = function(name) { return this[name]; };

for (i=0; i<10000; i++)
{
     if(getProperty("node_" +i).nodeNameType=="Enzyme")
     {
        //Rot
        getProperty("node_" +i).setFillColor(255,23,13,255);
     } else
     {
        if(getProperty("node_" +i).label=="CO2")
        {
          //Neongrün
          getProperty("node_" +i).setFillColor(10,233,133,255);
          getProperty("node_" +i).setHighlighted(true);
        }
        else
        {
          //Blau
          getProperty("node_" +i).setFillColor(10,3,233,255);
        }
     }
};
```

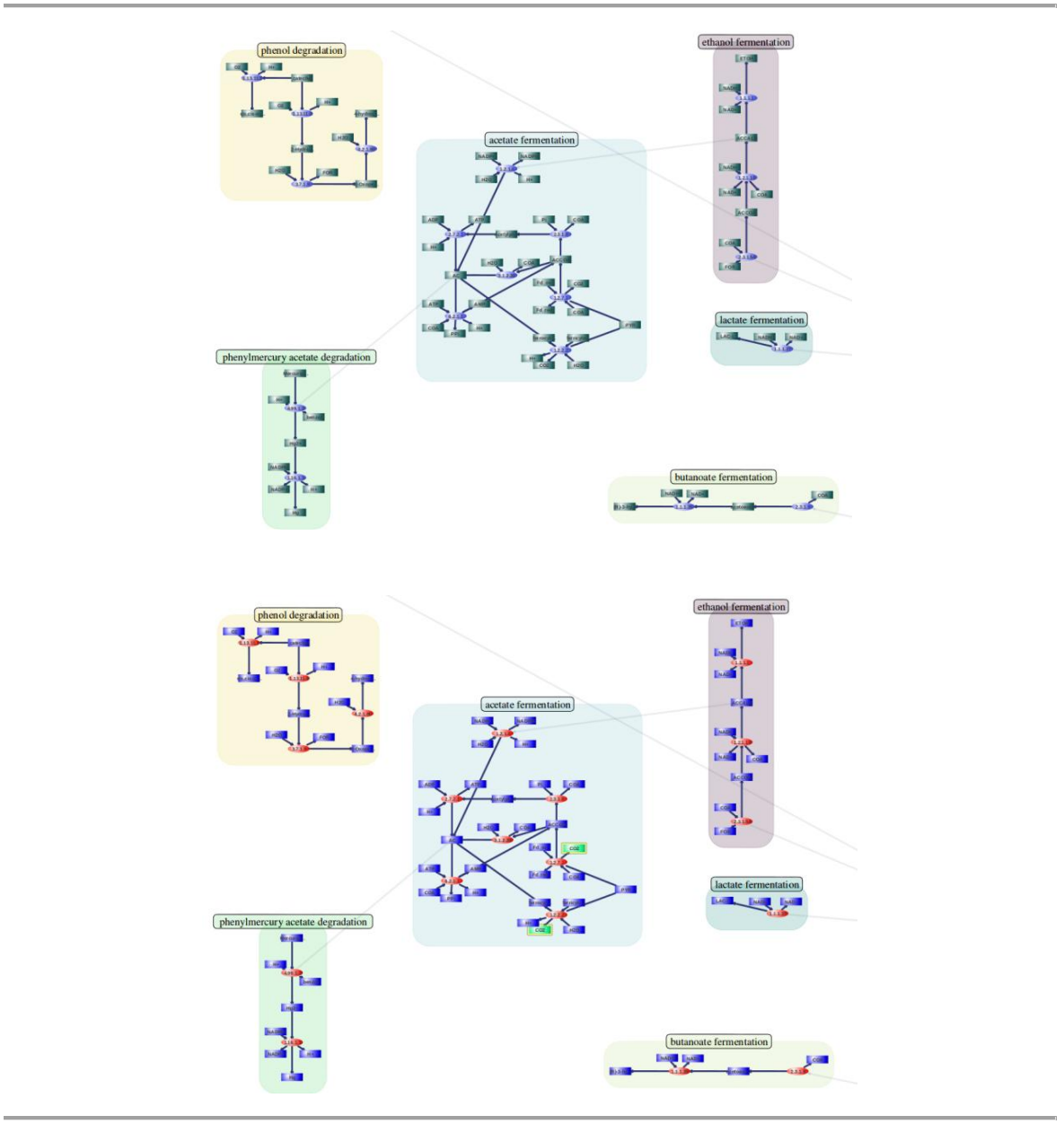

Abbildung 4.82: Auf dem oberen Bildausschnitt wird ein Teil der BRENDA-Maps im Ausgangszustand gezeigt. Im unteren Bildausschnitt wurden durch die Verwendung eines Skriptes (siehe Quellcode 4.2) Enzyme, Substanzen und Kohlendioxid im Speziellen farblich angepasst.

4.1.8 Vergleich mit anderen Systemen

In den vergangenen Jahren wurden einige Applikationen für die Darstellung metabolischer Netzwerke vorgestellt. Dieses Kapitel erhebt daher nicht den Anspruch, eine vollständige Übersicht über alle Pathway-Editoren zu bieten. Diese Arbeit beschränkt sich auf einen

detaillierten Vergleich mit den beiden Editoren, die weit verbreitet eingesetzt werden. Zunächst wird in Kapitel 4.1.8.1 die Softwaresuite Cytoscape vorgestellt, welche umfangreiche Analyse- und Darstellungsmöglichkeiten bietet. Kapitel 4.1.8.2 gibt eine kurze Einführung zu VANTED, welches die Besonderheit aufweist, ebenfalls Datenserien darstellen zu können. Abschließend folgt in Kapitel 4.1.8.3 ein Vergleich dieser zwei Programme mit MapOmnia.

4.1.8.1 Cytoscape

Cytoscape (Akira Funahashi *et al.*, 2003; Funahashi *et al.*, 2008; Shannon *et al.*, 2003; Cytoscape Consortium, 2013) ist eine umfangreiche Software-Applikation für die Darstellung und Manipulation metabolischer Netzwerke. Cytoscape ist in Java implementiert und dadurch als plattformunabhängig zu betrachten. Es steht unter der LGPL. Durch die Verwendung von Plugin-Schnittstellen bietet Cytoscape Entwicklern die Möglichkeit, modular weitere Funktionalitäten hinzuzufügen.

Die Softwaresuite unterstützt unter anderem das SIF-, das GML-, das XGMML- und das CSV-Format sowie den eigenen Dateityp CYS. Ebenfalls können Daten aus Datenbanken importiert werden. Die Version von Cytoscape, mit der dieser Vergleich durchgeführt wurde, ist 2.8.3. Weitere Funktionalitäten von Cytoscape sind in den Tabellen in Kapitel 4.1.8.3 gelistet. Abbildung 4.83 zeigt die Benutzeroberfläche (GUI) von Cytoscape.

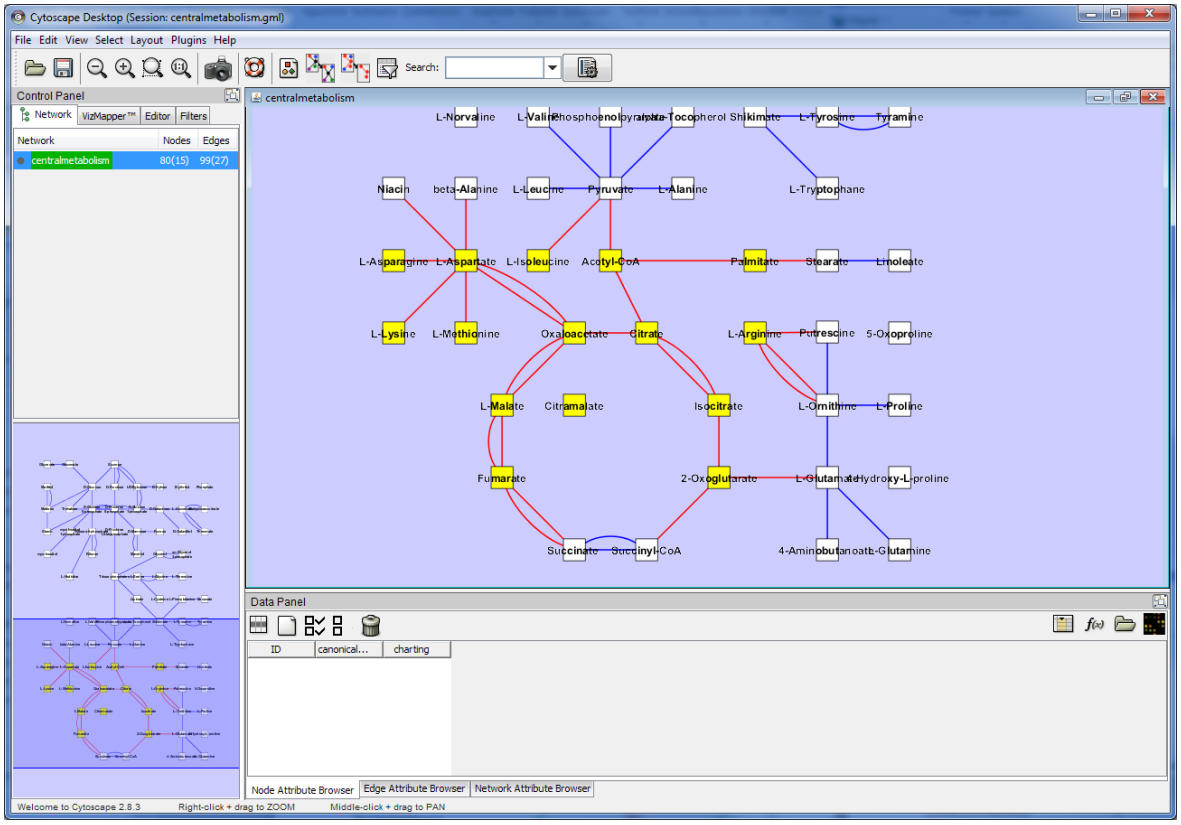

Abbildung 4.83: Benutzeroberfläche von Cytoscape: Die Bedienoberfläche bietet unter anderem neben der Netzwerkansicht, ein Andockfenster, um Daten der Netzwerkobjekte darzustellen, eine Übersichtskarte sowie eine Symbolleiste, um Programmfunktionen zu bedienen.

4.1.8.2 VANTED

VANTED (Junker *et al.*, 2006; Klukas, 2012) steht für „Visualization and Analysis of Networks containing Experimental Data“ und wurde am IPK Gatersleben entwickelt. Es unterstützt das Anzeigen, Analysieren und Erstellen von Netzwerken. Der Fokus dieser Software liegt im Visualisieren und Analysieren experimenteller Hochdurchsatz- (englisch: High-Throughput) Daten. VANTED ist in Java implementiert und dadurch als plattformunabhängig zu betrachten. Es unterstützt verschiedene Netzwerk-Austausch-Formate wie z. B. GML und SBML. Es können ebenfalls Netzwerke von KEGG im KGML-Format geladen werden. Die Version von VANTED, mit der dieser Vergleich durchgeführt wurde, ist V2.1.0. Eine detaillierte Auflistung von Funktionalitäten von

VANTED bieten die Tabellen in Kapitel 4.1.8.3. Abbildung 4.84 zeigt die Benutzeroberfläche von VANTED.

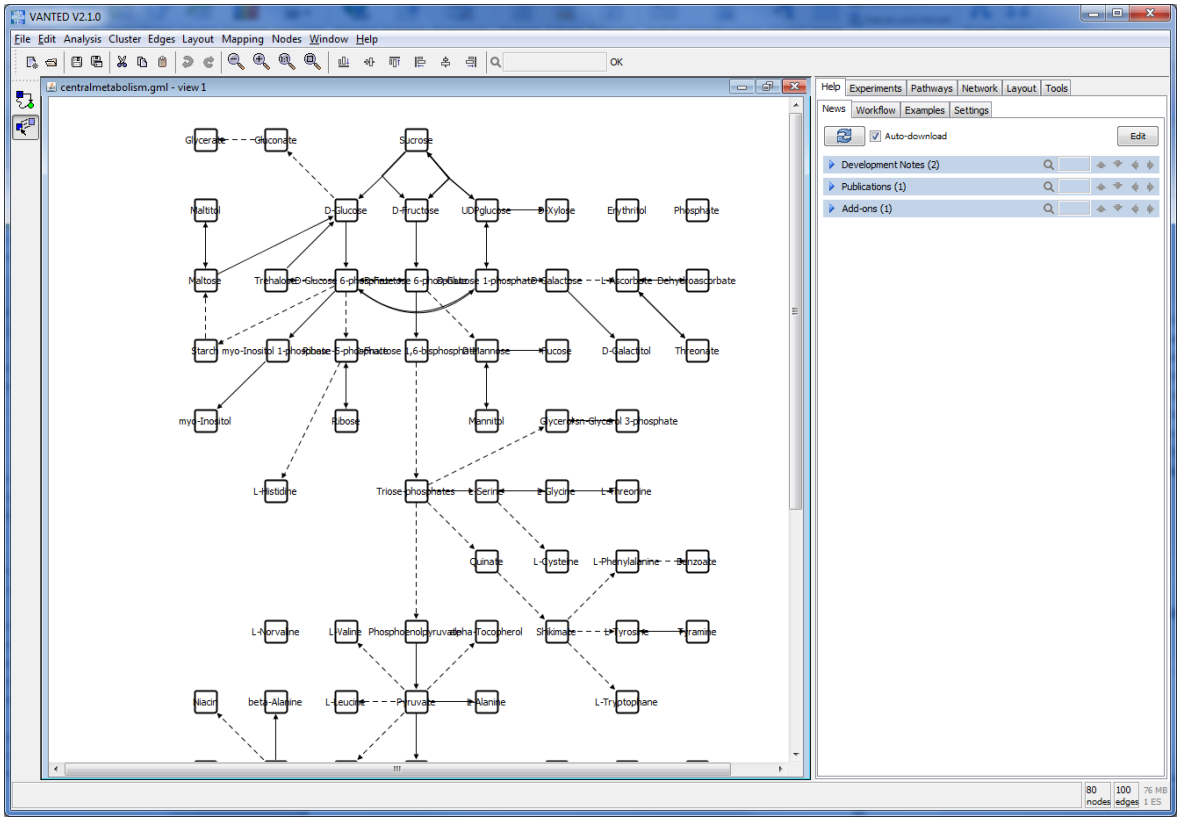

Abbildung 4.84: Benutzeroberfläche (GUI) von VANTED: Die Bedienoberfläche bietet dem Nutzer die Netzwerkansicht, in der er in dem Netzwerk navigieren kann. In dem rechten Fenster werden Programmfunktionen Tab-separiert angeboten. Im oberen Bereich kann der Nutzer über die Symbolleiste Funktionen für Navigation sowie Editierfunktionen bedienen.

4.1.8.3 Vergleich zu MapOmnia

Es ist anzumerken, dass die Softwarelösungen jeweils in der Basis-Version verglichen wurden. Das bedeutet, dass der Vergleich ohne Hinzuladen von Plugins durchgeführt wurde. Gerade bei Cytoscape werden, bedingt durch eine große Community von Entwicklern, etliche Plugins angeboten, welche zahlreiche zusätzliche Funktionalitäten offerieren.

Innerhalb des Vergleichs werden zunächst funktionale Komponenten der Pathway-Editoren gegenübergestellt (Unterabschnitt „Funktionale Komponenten“). Diese

Komponenten beschreiben unter anderem Funktionen wie Editiermöglichkeiten und schließen den strukturellen Aufbau der Frontends mit ein. Anschließend wird verglichen, wie performant die einzelnen Softwarelösungen sind (Unterabschnitt „Performance“). Zu diesem Zweck wurden Netzwerke unterschiedlicher Größe geladen, wobei Letztere die Ladezeiten und die Navigationsgeschwindigkeiten in diesen Netzwerken bestimmen. Die Darstellungsmöglichkeiten für Graphen werden in dem Unterabschnitt „Darstellung“ näher betrachtet und im Anschluss werden die Darstellungsmöglichkeiten der Netzwerkobjekte analysiert (Unterabschnitt „Darstellung der Netzwerkobjekte“). Methoden für die Abbildung experimenteller Daten werden in dem Unterabschnitt „Abbildung von Daten“ gegenübergestellt. In dem Unterabschnitt „Analyse-Möglichkeiten und Graphen-Algorithmik“ werden die Applikationen auf vorhandene Analysemethoden von Netzwerken untersucht. Hierzu gehören beispielsweise Korrelationsanalysen oder Graphsuchalgorithmen. Eine Untersuchung auf offerierte Operationen mit Graphen wie z. B. die Fusion zweier Knoten, wird im nachfolgenden Unterabschnitt „Operationen mit Graphen“ durchgeführt. Hier schließen sich die implementierten Layout-Funktionen der Applikationen an (Unterabschnitt „Layout-Funktionen“). Die Unterstützung der Import- und Export-Funktionalitäten unterschiedlicher Dateiformate wird in Unterabschnitt „Dateiformate“ ebenfalls gegenübergestellt. Es folgt in dem Unterabschnitt „Bildexport“ der Vergleich angebotener Bildexporte. Abschließend werden sonstige Merkmale der Editoren wie Lizenz, Kompatibilität oder Registrierung gegenübergestellt (Unterabschnitt „Lizenzmodelle und Betriebssysteme“).

Funktionale Komponenten

Tabelle 4.11 gibt eine Übersicht über die Basis-Funktionalitäten von Cytoscape, VANTED und MapOmnia. Allen Applikationen gemeinsam ist die Möglichkeit, die Bearbeitung von Netzwerken sowie umfangreiche Such- und Filteroptionen auf Basis ihrer Networkobjekte durchzuführen zu können. Eine Sortiermöglichkeit für Daten wird hingegen von VANTED nicht offeriert. Das Laden von experimentellen Daten wird wiederum von allen Softwareumsetzungen unterstützt, und sie verfügen ebenfalls über die Möglichkeit, erstellte Netzwerke als Bilddatei exportieren zu können.

Tabelle 4.11: Übersicht über die Basis-Funktionalitäten von Cytoscape, VANTED und MapOmnia

Funktionale Komponenten	Cytoscape	VANTED	MapOmnia
Editieren von Netzwerken	✓	✓	✓
Navigationsbaum für Netzwerke und Netwerkobjekte	✓	⊗	✓
Übersichtskarte	✓	⊗	✓
Abbildung experimenteller Daten	✓	✓	✓
Tooltip-Funktion	✓	✓	✓
Suchmöglichkeiten auf Basis von Netzwerkobjekten	✓	✓	✓
Filtermöglichkeiten auf Basis von Netzwerkobjekten	✓	✓	✓
Sortierfunktion von Attributen von Netzwerkobjekten	✓	⊗	✓
Export als Bilddatei	✓	✓	✓
Erweiterbarkeit über Plugin-Schnittstellen	✓	✓	✓
Skriptfähigkeit	⊗[1]	✓	✓
TCP/IP-Schnittstelle	⊗	⊗	✓
Delegate für Datenstrukturen	⊗	⊗	✓
Undo/Redo-Funktionalität	✓	✓	✓
Dynamische Attributverwaltung	✓	⊗	✓
Abbildung von Attribut auf Attribut	✓	✓	✓
Modellierungshilfe für die Erstellung neuer Netzwerke	✓	⊗	✓

Die Exportformate werden an späterer Stelle gegenübergestellt. Tooltip-Informationsfenster nutzen sowohl Cytoscape als auch VANTED und MapOmnia. Delegates für die Darstellung unterschiedlicher Datenstrukturen wird nur von MapOmnia

[1] nicht in der Basis-Version verfügbar, durch Plugin realisierbar

geboten. Die Umsetzung einer Undo/Redo-Funktionalität und die Abbildung von Attribut auf Attribut werden von allen Pathway-Editoren ermöglicht. Netzwerkobjekte dynamisch mit neuen Attributen zu erweitern, wird seitens VANTED nicht unterstützt. Ebenfalls findet sich keine Modellierungshilfe für die Konstruktion von Netzwerken in VANTED. Einen Navigationsbaum für Netzwerke und Netzwerkobjekte bieten nur Cytoscape und MapOmnia. Eine kleine Übersichtskarte wird auch nur von diesen beiden Applikationen angeboten.
Das Laden und Ausführen von Skripten kann in der Basis-Version ausschließlich in MapOmnia und VANTED erfolgen. Der Zugriff auf Netzwerke durch eine TCP/IP-Schnittstelle bietet nur MapOmnia.

Performance

Die Betrachtung der Performance unter den Aspekten von Ladezeit und dem Navigations- und Zoomverhalten wird eine zentrale Rolle für den Vergleich der Softwareapplikationen zugemessen. Auch Netzwerke mit einer großen Anzahl von Netzwerkobjekten (größer 10000) sollten von den Applikationen editier- und darstellbar sein. Tabelle 4.12 gibt eine Übersicht über die Kriterien für die Bestimmung performanter Unterschiede wieder. Für einen Vergleich wurden in VANTED randomisierte Netzwerke verschiedener Größe erstellt. Das erste umfasst 1.000 Knoten sowie 1.000 Kanten. Das zweite enthält jeweils 10-mal so viele Knoten und Kanten, das dritte 100-mal so viele. Es zeigen sich bei den Ladezeiten nur marginale Unterschiede zwischen den Applikationen. Es ist hierbei anzumerken, dass bei Cytoscape sowie MapOmnia zwischen der Ladezeit inklusive der Erstellung des Netzwerkes und der Zeit bis zu Visualisierung des Graphen differenziert wird. So war es zwar bei Cytoscape möglich das größte Netzwerk zu laden, allerdings konnte hierfür keine Ansicht generiert werden.
Bei dem Vergleich der Navigation innerhalb der unterschiedlich großen Netzwerke zeigt sich, dass MapOmnia das performanteste Verhalten aufweist. Cytoscape war nicht in der Lage, eine Ansicht des großen Netzwerks zu generieren. Eine Rotation der Ansicht auf das Netzwerk wird ausschließlich von MapOmnia offeriert.

Tabelle 4.12: Übersicht über die Kriterien für die Bestimmung performanter Unterschiede zwischen Cytoscape, VANTED und MapOmnia. Die Laufzeitmessungen wurden jeweils dreimalig durchgeführt. Die Standardabweichung ist in Klammern angegeben. Der Tabelleneintrag „flüssig“ zeigt an, dass keine Verzögerungen messbar waren. Mit ⊗ markierte Tabelleneinträge geben an, dass diese Funktion nicht von der Applikation unterstützt wird

Performance		**Cytoscape**	**VANTED**	**MapOmnia**
Ladezeit	1000 Knoten und Kanten	**2 (±0,2) sec**	**2 (±0,2) sec**	**0,5 (±0,1) sec**
	10000 Knoten und Kanten	**6 (±0,4) sec**	**10 (±0,6) sec**	**4 (±0,2) sec**
	100000 Knoten und Kanten	**50 (±4) sec**	**80 (±5) sec**	**45 (±3) sec**
1000 Knoten und Kanten	Zoom	**flüssig**	**flüssig**	**flüssig**
	Navigation	**flüssig**	**flüssig**	**flüssig**
	Rotation der Karte	⊗	⊗	**flüssig**
10000 Knoten und Kanten	Zoom	**flüssig**	**<1 sec**	**flüssig**
	Navigation	**flüssig**	**<1 sec**	**flüssig**
	Rotation der Karte	⊗	⊗	**flüssig**
100000 Knoten und Kanten	Zoom	**nicht durchführbar**	**5 (±0,4) sec**	**<1 sec**
	Navigation	**nicht durchführbar**	**3 (±0,3) sec**	**<1 sec**
	Rotation der Karte	⊗	⊗	**<1 sec**

Darstellung

Die Darstellung von Netzwerken (siehe Tabelle 4.13) unterscheidet sich nicht wesentlich innerhalb der drei Pathway-Editoren. Alle drei Softwarelösungen können eine Gesamtkarte sowie Teilgraphen in der zweidimensionalen Ebene darstellen. Die Darstellung von Kantenrichtungen dargestellt als Pfeile wird ebenfalls von den Programmen unterstützt.

Tabelle 4.13: Übersicht über die Darstellung der Netzwerke in Cytoscape, VANTED und MapOmnia

Darstellung	Cytoscape	VANTED	MapOmnia
Bipartiter Graph	✓	✓	✓
Gesamtkarte	✓	✓	✓
Teilgraphen	✓	✓	✓
Kantenrichtungen dargestellt als Pfeile	✓	✓	✓
Darstellung von Diagrammen	⊗	✓	⊗
Darstellung von Zeitseriendaten als Diagramme	⊗	✓	⊗
Darstellung von Hintergrundbilder in der Netzwerkszene	⊗	✓	✓
Moleküldarstellung	⊗	⊗	✓
2D Darstellung von Netzwerken	✓	✓	✓
3D Darstellung von Netzwerken	⊗	⊗	⊗
Netzwerkobjekt „Gruppe“ Gruppierung von Knoten und Kanten	⊗[2]	⊗	✓
Animation der Netzwerkszene	⊗	⊗	✓
Animation der Netzwerkobjekte	⊗[2]	⊗	✓
zoomabhängiges Darstellungsverhalten	⊗	⊗	✓

[2] nicht in der Basis-Version verfügbar, durch Plugin realisierbar

Die Darstellung von Molekülstrukturen unterstützt nur MapOmnia. VANTED ist als einziges Programm in der Lage, Zeitseriendaten als Diagramme zu visualisieren. MapOmnia hingegen bietet als einzige Applikation das Netzwerkobjekt „Gruppe“ neben Knoten und Kanten. Ein weiteres Alleinstellungsmerkmal von MapOmnia sind die Darstellungen von Animationen der Netzwerkszene sowie der Netzwerkobjekte. Auch das zoomabhängige Darstellungsverhalten wird nicht von den anderen Systemen verwendet. Hierbei werden die Netzwerkobjekte von der Detailstufe abhängig dargestellt.

Darstellung der Netzwerkobjekte

Die Festlegung grundlegender Eigenschaften der Netzwerkobjekte (siehe Tabelle 4.14) wie Rahmenfarbe, Rahmenstärke, Füllfarbe oder Beschriftung wird von allen Pathway-Editoren unterstützt. Die Verwendung von Mustern für Rahmen und Füllfläche der Netzwerkobjekte sowie die Verwendung von Gradienten bieten hingegen nur VANTED und MapOmnia an. VANTED besitzt mit 22 verschiedenen Knotentypen die höchste Auswahl. Die Darstellung von dynamisch nachgeladenen Bilddateien innerhalb der Knoten wird nur von VANTED und MapOmnia unterstützt, die Darstellung von Datenplots ausschließlich von VANTED.

Tabelle 4.14: Übersicht über die Darstellung der Netzwerkobjekte in Cytoscape, VANTED und MapOmnia

Darstellung Netzwerkobjekte	Cytoscape	VANTED	MapOmnia
Rahmenfarbe	✓	✓	✓
Rahmenstärke	✓	✓	✓
Muster für Umrahmung	⊗	✓	✓
Füllfarbe	✓	✓	✓
Gradientenfüllung	⊗	✓	✓
Muster für Füllung	⊗	✓	✓
Anzahl geometrischer Formen	9	22	15
Pfaddarstellung	⊗	⊗	✓
Beschriftung der Netzwerkobjekte	✓	✓	✓
Darstellung von Bildern innerhalb der Netzwerkobjekte	⊗	✓	✓
Darstellung von gekapselten Netzwerken innerhalb der Netzwerkobjekte	⊗	⊗	⊗
Darstellung von Datendiagrammen innerhalb der Netzwerkobjekte	⊗	✓	⊗

Abbildung von Daten

Tabelle 4.15 gibt eine Übersicht über die angebotenen Datenabbildungen in Cytoscape, VANTED und MapOmnia. Jeder der vorgestellten Pathway-Editoren bietet die

Möglichkeit, Daten auf ein bestehendes Netzwerk abzubilden. Hierbei werden eine Größencodierung und eine Farbcodierung unterstützt.

Tabelle 4.15: Übersicht über die angebotenen Datenabbildungen in Cytoscape, VANTED und MapOmnia

Abbildung von Daten	Cytoscape	VANTED	MapOmnia
Farbcodierung	✓	✓	✓
Größencodierung	✓	✓	✓

Analyse-Möglichkeiten und Graphen-Algorithmik

Die Berechnung kürzester Wege in einem vorhandenen Netzwerk wird von VANTED und MapOmnia unterstützt. Die Durchführung von t-Tests, Korrelationsanalysen sowie der Visualisierung von Korrelationsberechnungen im Netzwerk ausschließlich von VANTED. Tabelle 4.16 listet die Analyseoptionen in Cytoscape, VANTED und MapOmnia.

Tabelle 4.16: Übersicht von Analyseoptionen in Cytoscape, VANTED und MapOmnia

Analyse	Cytoscape	VANTED	MapOmnia
Kürzeste Wege	⊗	✓	✓
T-Test	⊗	✓	⊗
Korrelationsanalyse	⊗	✓	⊗
Visualisierung von Korrelationsberechnungen	⊗	✓	⊗

Operationen mit Graphen

Das Entfernen von Knoten und Kanten wird von allen drei Pathway-Editoren unterstützt, die Fusion zweier Knoten zu einem hingegen nur von VANTED und MapOmnia. Eine Kontraktion zweier Kanten bietet ausschließlich die Softwaresuite MapOmnia. Cytoscape sowie MapOmnia erlauben den Schnitt und die Vereinigung zweier Graphen. Diese Funktion ist in VANTED nicht verfügbar. Tabelle 4.17 gibt eine Übersicht über die angebotenen Operationen mit Graphen in Cytoscape, VANTED und MapOmnia.

Tabelle 4.17: Übersicht über die angebotenen Operationen mit Graphen in Cytoscape, VANTED und MapOmnia

Operationen mit Graphen	Cytoscape	VANTED	MapOmnia
Entfernen von Knoten und Kanten	✓	✓	✓
Fusion von Knoten	⊗	✓	✓
Kontraktion von Kanten	⊗	⊗	✓
Vereinigung zweier Graphen	✓	⊗	✓
Schnitt zweier Graphen	✓	⊗	✓

Layout-Funktionen

Cytoscape, VANTED und MapOmnia bieten mehrere Layout-Funktionen an (siehe Tabelle 4.18). Alle Applikationen unterstützen die Layout-Methoden für eine lineare, eine rasterorientierte sowie eine randomisierte Anordnung der Knoten. Auch kräftebasiertes Layouting lässt sich in allen drei Editoren durchführen. VANTED bietet zirkuläres Layouting an, jedoch ohne elliptische Ausrichtungsoption. Eine Anordnung entlang von Schrifttexten bietet nur MapOmnia. Das Verschieben auf nächste Nachbarpunkte von ausgewählten Netzwerknoten unterstützen nur VANTED und MapOmnia. Eine Skalierung und Spiegelung von Knoten kann sowohl in Cytoscape als auch in MapOmnia erfolgen. Eine Ausrichtung in Form eines Tree-Layouts wird von Cytoscape und VANTED unterstützt.

Dateiformate

Tabelle 4.19 listet insgesamt 18 verschiedene Netzwerkdateiformate. Von diesen können sowohl Cytoscape als auch VANTED jeweils acht laden. MapOmnia verfügt über Import-Möglichkeiten für sechs Dateiformate. VANTED bietet als einziger Pathway-Editor den Import von GraphML, PAJEK-NET sowie DOT-File an. Cytoscape ist als einziges Programm in der Lage, NNF sowie PSI-MI Dateien zu laden. Möchte der Betrachter molekulare Strukturinformationen importieren, so ist dieses nur mit MapOmnia möglich. Die Dateiformate CellML, SBGN, XWG, INA PNT und DAT Metatool kann keines der Applikationen importieren.

Tabelle 4.18: Übersicht über die Layout-Funktionen in Cytoscape, VANTED und MapOmnia

Layout-Funktionen	Cytoscape	VANTED	MapOmnia
Lineare Anordnung („Linear Layout“)	✓	✓	✓
Rasterorientierte Anordnung („Grid Layout“)	✓	✓	✓
Elliptische Anordnung („Elliptical Layout“)	✓	⊗	✓
Anordnung entlang von Schrifttexten („Painter Path Layout“)	⊗	⊗	✓
Kräftebasierte Anordnung („Force-Based Layout“)	✓	✓	✓
Randomisierte Anordnung („Random“)	✓	✓	✓
Baum-basierte Anordnung („Tree Layout“)	✓	✓	⊗
Rotation	✓	✓	✓
Skalierung	✓	⊗	✓
Spiegelung	✓	⊗	✓
Verschieben auf nächste Nachbarpunkte	⊗	✓	✓

Tabelle 4.19: Übersicht über Netzwerkimport-Formaten in Cytoscape, VANTED und MapOmnia

Import	Cytoscape	VANTED	MapOmnia
GraphML	⊗	✓	⊗
CellML	⊗	⊗	⊗
XGMML	✓	⊗	✓
KGML	⊗	✓	✓
SBML	✓	✓	✓
GML	✓	✓	✓
CSV	✓	✓	✓
MDL Molfile	⊗	⊗	✓
SBGN	⊗	⊗	⊗
PAJEK NET	⊗	✓	⊗
BioPax OWL	✓	✓	⊗
PSI-MI	✓	⊗	⊗
SIF	✓	✓	⊗
NNF	✓	⊗	⊗
DOT	⊗	✓	⊗
XWG	⊗	⊗	⊗
INA PNT	⊗	⊗	⊗
DAT Metatool	⊗	⊗	⊗

Bei dem Export von Dateien (siehe Tabelle 4.20) unterstützt MapOmnia lediglich die Formate CSV und GML. VANTED bietet mit 11 unterstützten Formaten umfangreichste Kompatibilität an. Cytoscape kann insgesamt 8 der 18 Netzwerkformate schreiben. Von keiner der drei Applikationen wird das Dateiformat CellML, KGML, MDL Molfile und SBGN unterstützt.

Tabelle 4.20: Übersicht über Netzwerkexport-Formaten in Cytoscape, VANTED und MapOmnia

Export	Cytoscape	VANTED	MapOmnia
GraphML	⊗	✓	⊗
CellML	⊗	⊗	⊗
XGMML	✓	⊗	⊗
KGML	⊗	⊗	⊗
SBML	✓	✓	⊗
GML	✓	✓	✓
CSV	✓	✓	✓
MDL Molfile	⊗	⊗	⊗
SBGN	⊗	⊗	⊗
PAJEK NET	⊗	✓	⊗
BioPax OWL	✓	✓	⊗
PSI-MI	✓	⊗	⊗
SIF	✓	✓	⊗
NNF	✓	⊗	⊗
DOT	⊗	✓	⊗
XWG	⊗	✓	⊗
INA PNT	⊗	✓	⊗
DAT Metatool	⊗	✓	⊗

Bildexport

Durch die Verwendung der von Qt bereitgestellten Rendermöglichkeiten bietet MapOmnia Unterstützung der gängigsten Bildformate. So kann MapOmnia insgesamt 13 der in der Tabelle 4.21 gelisteten Bildformate generieren. Der Exportumfang von Cytoscape und VANTED ist mit sieben bzw. vier unterstützten Formaten geringer.

Tabelle 4.21: Übersicht offerierter Exportformate für Bilder in Cytoscape, VANTED und MapOmnia

Bildexport	**Cytoscape**	**VANTED**	**MapOmnia**
BMP	✓	⊗	✓
GIF	⊗	⊗	✓
JPG	✓	✓	✓
JPEG	✓	⊗	✓
PNG	✓	✓	✓
PBM	⊗	⊗	✓
PGM	⊗	⊗	✓
PPM	⊗	⊗	✓
TIFF	⊗	⊗	✓
XBM	⊗	⊗	✓
XPM	⊗	⊗	✓
PDF	✓	✓	✓
SVG	✓	✓	✓
EPS	✓	⊗	⊗

Lizenzmodelle und Betriebssysteme

Der Vergleich der in diesem Abschnitt beschriebenen Merkmale betrifft die Kompatibilität, das Lizenzsystem sowie Nutzbarkeit der Applikationen (siehe Tabelle

4.22). Jeder der drei Pathway-Editoren ist sowohl unter Windows als auch unter Linux nutzbar. MapOmnia wurde ebenfalls für Windows kompiliert. Sämtlich verwendete Bibliotheken weisen die nötige Cross-Kompatibilität auf. Das Lizenzmodell ist für Cytoscape sowie MapOmnia die *GNU General Public*-Lizenz und für VANTED die *GNU Lesser General Public*-Lizenz. Jede Applikation ist frei nutzbar, und für die Benutzung ist keine Registrierung erforderlich.

Tabelle 4.22: Übersicht über Lizenzmodelle und unterstützte Betriebssysteme von Cytoscape, VANTED und MapOmnia

Lizenzmodelle **Betriebssysteme**	**Cytoscape**	**VANTED**	**MapOmnia**
Lizenz	LGPL	GPL	LGPL
Windows kompatibel	✓	✓	✓[3]
Mac kompatibel	✓	✓	⊗
Linux kompatibel	✓	✓	✓
Freie Benutzung	✓	✓	✓
Registrierung	⊗	⊗	⊗

4.2 MapNet-Server

Um das Webprojekt BRIME (Kapitel 4.3) zu verwirklichen, wurde ein eigener TCP/IP-Server implementiert. Bei BRIME handelt es sich um einen webbasierten Zugang zu Stoffwechselkarten. Der MapNet-Server kann die Funktionalität von MapOmnia (Kapitel 4.1) benutzen, um Netzwerke unterschiedlichen Ursprungs rendern zu lassen. Hierbei ist eine Schnittstelle entstanden, welche in der Lage ist, Anfragen von Clients zu verarbeiten und diese zu beantworten.

[3] Die Kompilierung unter Windows wurde ebenfalls durchgeführt.

4.2.1 Programmmetrik des MapNet-Servers

Bei der Entwicklung des MapNet-Servers konnten Klassen des Programms MapOmnia wiederverwendet werden. Insbesondere Klassen, die für die Datenstruktur sowie für die Darstellung der Netzwerke implementiert wurden, finden im MapNet-Projekt Verwendung. Es mussten einige wiederverwendete Klassen für die serverseitige Verwendung optimiert bzw. angepasst werden, um den nötigen Anforderungen wie z. B. der hohen Performance gerecht zu werden.

MapNet besitzt 80 verschiedene Klassen und über 1.300 verschiedene Funktionen. Somit kommen auf eine Klasse im Durchschnitt etwa 16 Funktionen. Die Anzahl der Quelltextzeilen beträgt ca. 14.000, was durchschnittlich 18 Zeilen Code pro Funktion bedeutet. Tabelle 4.23 listet die genauen Metriken.

Tabelle 4.23: Übersicht der Programmmetrik des MapNet-Servers

Typ	Anzahl
Dateien	100
Klassen	80
Funktionen	1.305
Zeilen gesamt	22.086
Quelltextzeilen	13.898
Kommentarzeilen	3.655
Leerzeilen	3.781
Deklarative Anweisungen	4.878
Ausführbare Anweisungen	5.281
Verhältnis: Funktionen pro Klasse	16
Verhältnis: Quelltextzeilen pro Funktion	11
Verhältnis: Kommentarzeilen zu Quelltextzeilen	0,26

4.2.2 Quelltext-Dokumentation des MapNet-Servers

Wie auch die Dokumentation des Programmes MapOmnia erfolgte die Dokumentation von MapNet mit dem unter 3.6.4 beschriebenen Programm Doxygen. Abbildung 4.85 zeigt den Klassenindex der erstellten HTML-Dokumentation.

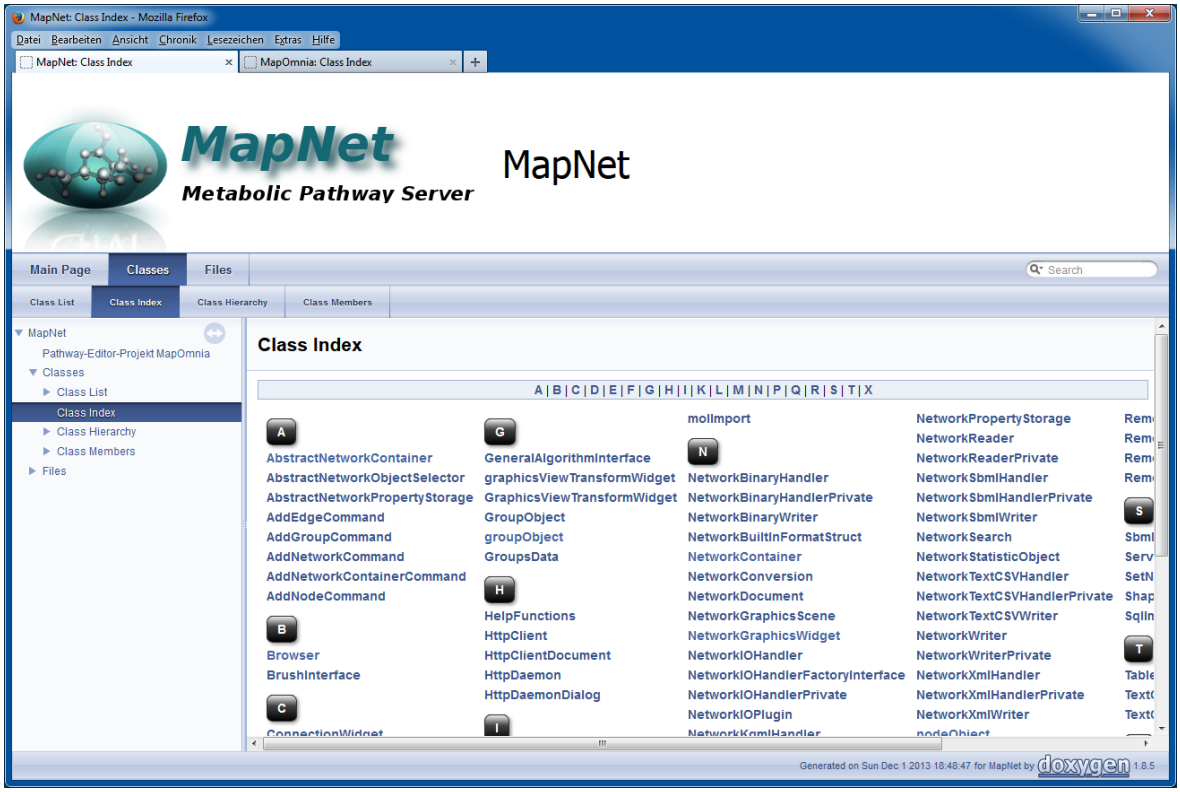

Abbildung 4.85: Die Doxygen-Dokumentation bietet eine strukturierte Übersicht über den entwickelten MapNet-Server. Die obige Ansicht zeigt den Klassenindex. Hier sind sämtliche Klassen alphabetisch angeordnet und zu den jeweiligen Klassenbeschreibungen verlinkt. Die vollständige Doxygen-Dokumentation befindet sich auf der zu dieser Arbeit beigefügten Daten-DVD-Rom.

4.2.3 Klassen des MapNet-Servers

In den folgenden Kapiteln werden die implementierten Klassen des TCP/IP-Servers beschrieben. Kapitel 4.2.3.1 widmet sich dem Dialogfenster „HttpDaemonDialog“. Hierauf folgt eine Beschreibung der Implementierung des TCP-Servers *HttpDaemon* (Kapitel 4.2.3.2). In Kapitel 4.2.3.3 wird die Klasse *HttpClientDocument*, welches das Netzwerk

verwaltet, vorgestellt. Das Kapitel 4.2.3.4 behandelt abschließend den *HttpClient*, welcher die benutzerspezifischen Daten der Clients verwaltet.

4.2.3.1 “HTTP-Daemon“ Dialogfenster

Der Start des Servers erfolgt GUI-basiert (siehe Abbildung 4.86). Innerhalb des sich öffnenden Dialogfensters „HTTP-Daemon“ kann der Port festgelegt sowie der Server gestartet, gestoppt und pausiert werden. Um die Anwendung performant zu halten, kann eine Zeit vorgegeben werden, nach der untätige TCP/IP-Verbindungen zu Clients automatisch geschlossen werden.

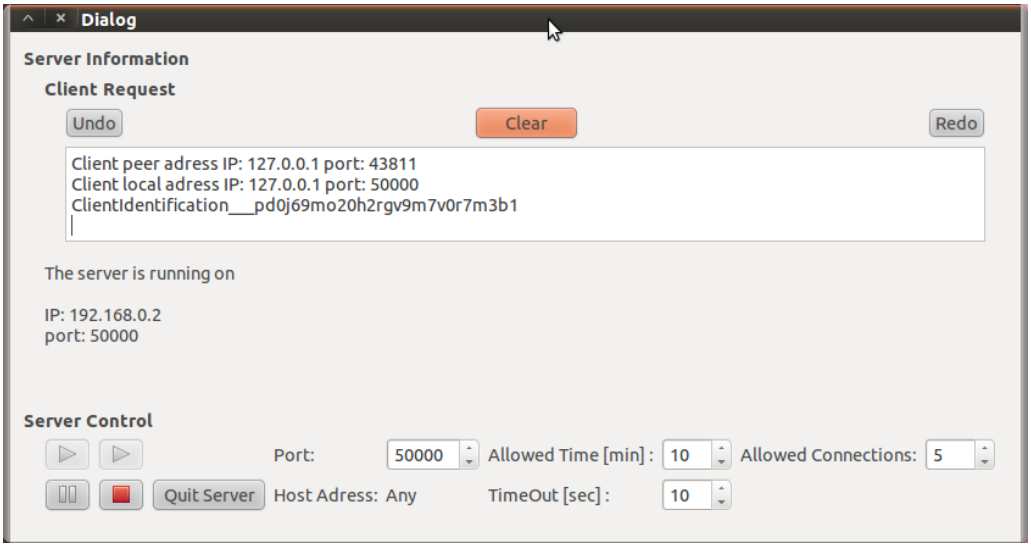

Abbildung 4.86: Bedienoberfläche des „HTTP-Dämon“. Das GUI-basierte Frontend bietet dem Nutzer verschiedene Konfigurationsmöglichkeiten. Der Dämon kann gestartet, pausiert sowie gestoppt werden. Auch lässt sich der Port festlegen, der für die Kommunikation zu den Clients genutzt wird. Ein integrierter Textbrowser stellt die clientseitigen Anfragen chronologisch sortiert dar.

Beim Start des Servers wird je ein Objekt der Klasse *HttpDaemon* sowie *HttpClientDocument* erstellt, und die Kommunikation zwischen diesen und dem *HttpDaemonDialog* aufgebaut. Anfragen von neu erstellten TCP-Sockets werden innerhalb dieser Klasse an das *HttpClientDocument*-Objekt weitergeleitet, das seinerseits die Verwaltung der *HttpClient*-Objekte übernimmt.

4.2.3.2 Klasse *HttpDaemon*

Die Klasse *HttpDaemon* stellt den TCP-Server und ist abgeleitet von *QTcpServer*, der Basisklasse für TCP-Server von Qt. Dieser nimmt Anfragen von Clients entgegen und gibt anschließend die berechneten Daten des Programmes an die Clients zurück. Für jede neue eingehende Verbindung wird ein eigener TCP-Socket erstellt.

4.2.3.3 Klasse *HttpClientDocument*

Die Klasse *HttpClientDocument* dient der Verwaltung des *NetworkDocuments* (siehe Kapitel 4.1.4.2) und der Erzeugung sowie Verwaltung von Objekten der Klasse *HttpClient*. Zu der Verwaltung gehört ebenfalls das zeitgesteuerte Entfernen und Löschen von untätigen Client-Verbindungen.

4.2.3.4 Klasse *HttpClient*

Für jede neue TCP-Verbindung wird ein Objekt der Klasse *HttpClient* erstellt. Es dient als individuelle Schnittstelle für die eingehenden Kommunikationsbefehle. Jedes erzeugte Objekt besitzt einen Pointer auf das erstellte *NetworkDocument* (siehe Kapitel 4.1.4.2), um die Manipulationen auf dem erstellten Netzwerk ausführen zu können. Bei der Erstellung eines Objekts der Klasse *HttpClient* werden vier Objekte der Klasse *NetworkContainer* (siehe Kapitel 4.1.4.2) erzeugt, wobei drei die Aufgabe übernehmen, benutzerspezifische Teilnetzwerke darzustellen. Das Vierte dient der netzwerkbasierten Speicherung von Suchanfragen.

Innerhalb der *setControl()*-Methode der Klasse des *HttpClient* wird die Anfrage eines Clients verarbeitet und der transferierte Befehl ausgeführt. Hierfür sind die Methoden für die Befehlsausführung innerhalb dieser Klasse zusätzlich gekapselt und werden an den finalen Empfänger, das erstellte Objekt der Klasse *NetworkDocument*, weitergereicht. Die Klasse speichert bei jeder neuen Anfrage durch den Client einen Zeitstempel, um eine zeitgesteuerte Löschung der Instanz des *HttpClient* vornehmen zu können. Dieses Verfahren sorgt dafür, den Speicherplatz-Bedarf gering zu halten.

4.2.4 Kommunikation

Der TCP-basierte Server nutzt das in Kapitel 3.5 vorgestellte Transmission-Control-Protocol. Hierbei wartet der Server auf allen Adressen seines Hosts auf Anfragen, wobei ebenfalls ein spezifischer Port bestimmt werden kann. Sofern eine Anfrage eines Clients eintritt, wird ein TCP-Socket erstellt, der für die Kommunikation mit diesem Client verwendet wird. Hierbei werden die Datenpakete ausgetauscht und entsprechend verarbeitet. Der TCP-Server unterdessen wartet auf weitere Anfragen. Abbildung 4.87 skizziert dieses Verhalten.

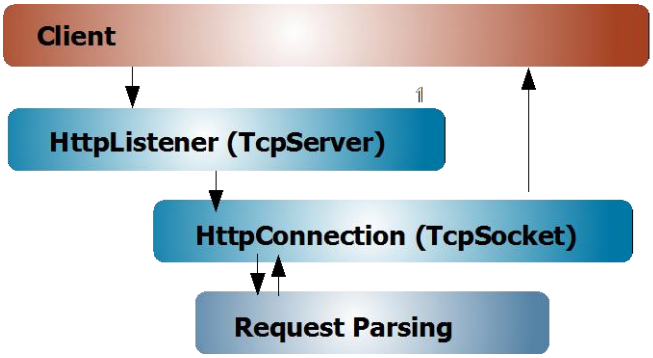

Abbildung 4.87: Die Grafik zeigt schematisch, wie eine Anfrage eines Clients dazu führt, dass der TCP-Server eine Kommunikationsbrücke durch Erstellen eines TCP-Sockets ermöglicht. Der Socket verarbeitet hierbei die Anfragen.

4.3 Webprojekt BRIME

In diesem Kapitel wird die Konzeption und die Realisierung des Webprojekts BRIME vorgestellt. Die Präsentation der Ergebnisse erfolgt analog dem MapOmnia Projekt. Zunächst werden in Kapitel 4.3.1 die statistischen Parameter der Programmmetrik von BRIME aufgezeigt. Kapitel 4.3.2 verweist auf die Dokumentation. In Kapitel 4.3.3 wird das Kommunikationsnetzwerk beschrieben und Kapitel 4.3.4 zeigt die Benutzeroberflächen der Webapplikation. Die Komponenten und die Bedienung von BRIME werden detailliert in Kapitel 4.3.5 vorgestellt. Die grafischen Anpassungsmöglichkeiten von Netzwerkobjekten unterscheiden sich grundlegend von denen in MapOmnia; diese werden in Kapitel 4.3.6 aufgezeigt. Nach den theoretischen

Aspekten folgen auch hier Anwendungsbeispiele (Kapitel 4.3.7). In Kapitel 4.3.8 wird BRIME mit anderen Web-Browsern von metabolischen Netzwerken verglichen.

4.3.1 Programmmetrik von BRIME

Bei BRIME handelt es sich um ein Softwareprojekt geringeren Umfangs im Vergleich zu MapOmnia, sofern der entwickelte Server MapNet nicht in der Statistik der Programmetrik berücksichtigt wird. BRIME besitzt 16 Klassen und über 450 verschiedene Funktionen. Die Anzahl der Zeilen Quelltext ist mit über 7500 zu betiteln. In Tabelle 4.24 sind die genauen Metrik-Zahlen aufgelistet.

Tabelle 4.24: Übersicht der Programmmetrik von BRIME

Typ	Anzahl
Dateien	76
Klassen	16
Funktionen	458
Zeilen gesamt	13.767
Quelltextzeilen	7.546
Kommentarzeilen	2.104
Leerzeilen	2.529
Deklarative Anweisungen	1.104
Ausführbare Anweisungen	5.291
Verhältnis: Funktionen pro Klasse	41
Verhältnis: Quelltextzeilen pro Funktion	16
Verhältnis: Kommentarzeilen zu Quelltextzeilen	0,28

4.3.2 Dokumentation von BRIME

Für die Webpräsenz BRIME wird auf der Webseite eine umfangreiche Hilfeseite angeboten. Hier werden im Einzelnen die Komponenten sowie die Bedienung erörtert. Ebenfalls bietet diese Hilfeseite zahlreiche Anwendungsbeispiele, um den Umgang mit BRIME zu erlernen.

4.3.3 Kommunikation

Abbildung 4.88 gibt eine Übersicht über das Kommunikationsnetzwerk. Bei Start des C++-Servers wird eine Verbindung zu der Datenbank „metabolic_pathways“ (siehe Kapitel 3.9.5) aufgebaut, wodurch der Server in der Lage ist, die nötigen Netzwerkdaten zu erhalten, um das Netzwerk für die Visualisierung erstellen zu können. Kommen Anfragen eines Clients über einen Apache-Webserver, baut PHP zu dem C++-Server eine Verbindung auf, wobei diese über eine neu erstellte Socket-Schnittstelle kommunizieren. Der C++-Server verarbeitet die spezifischen Anfragen des Clients. Hierbei kann die Anfrage ebenso die Kartennavigation als auch das Abbilden von Daten betreffen. Anschließend gibt der Server das gerenderte Bild im PNG-Format sowie in der gewünschten Größe und Kartenposition zurück. Gleichzeitig erstellt der C++-Server eine ImageMap (deutsch: die verweissensitive Grafik), welche dem Client übermittelt wird. Für die Identifizierung von Komponenten aus der BRENDA-Maps kommuniziert PHP mit dem in Kapitel 3.9.6 vorgestellten Python Tool „ReMeRe“ von Melanie Busch.

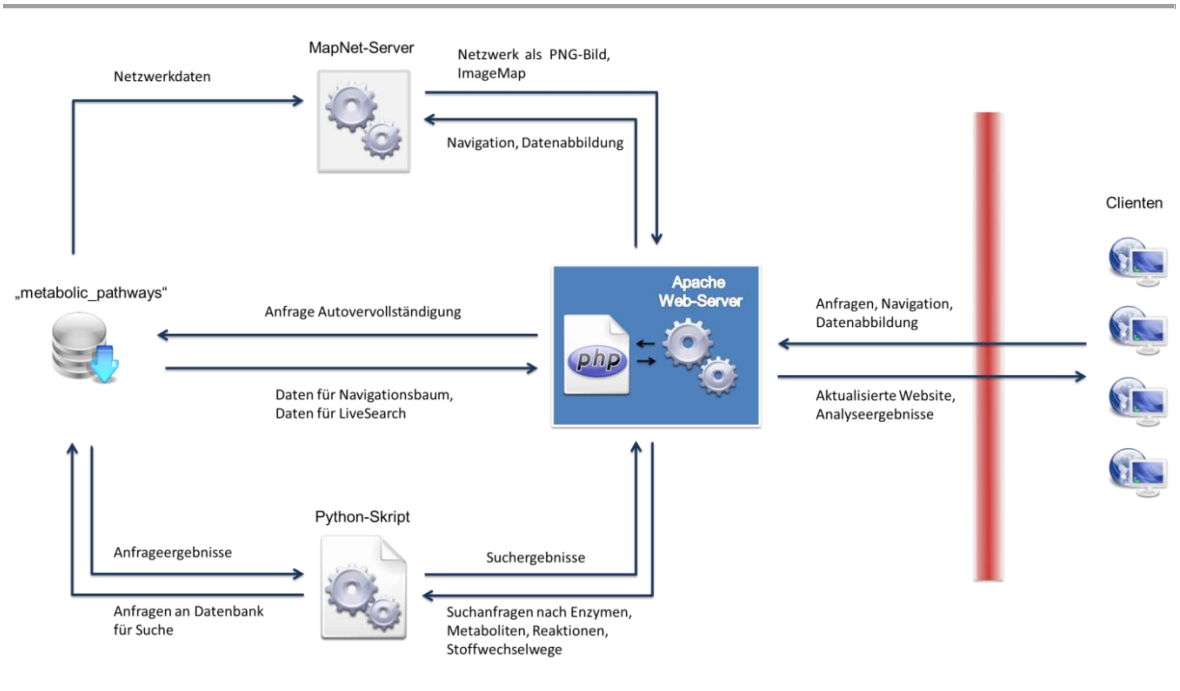

Abbildung 4.88: Für den Aufbau und die Navigation innerhalb der Stoffwechselkarte in BRIME ist das Kommunikationsnetzwerk schematisch dargestellt. Die in der Grafik verwendeten Symbolbilder stehen unter der LGPL-Lizenz und wurden von Everaldo Coelho (http://www.everaldo.com/) erstellt.

4.3.4 Benutzeroberfläche von BRIME

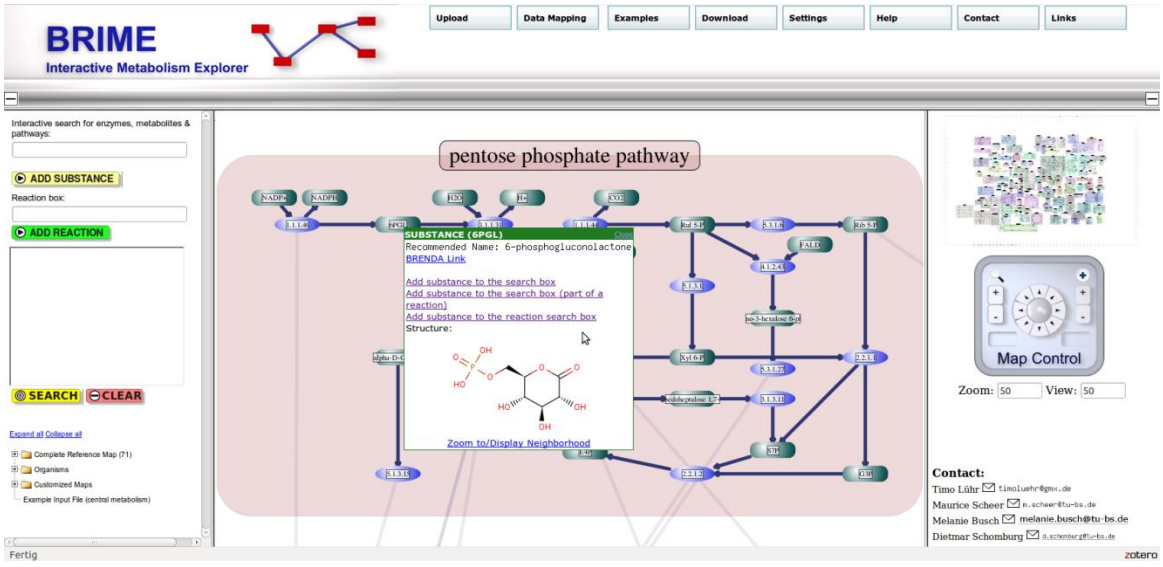

Abbildung 4.89: Gestaltung der BRIME Web-Page: Die Webseite ist in vier Frames aufgeteilt. Im oberen Frame ist das Menü lokalisiert. Im linken Frame befindet sich der Auswahlbereich. Im mittleren Frame wird das Netzwerk dargestellt und im rechten Frame gibt es zusätzliche Navigationsmöglichkeiten sowie eine Übersichtskarte.

In Abbildung 4.89 ist die Weboberfläche der BRIME-Homepage dargestellt. Die Webseite wird durch ein Frameset (deutsch: das Rahmen-Set) in vier Bereiche geteilt. Hierdurch hat der Benutzer die Möglichkeit, durch eigene Größenanpassung der Frames die Seite nach seinen Wünschen zu gestalten. Der strukturelle Aufbau ähnelt der Benutzeroberfläche des Pathway-Editors MapOmnia, wobei der Funktionsumfang deutlich reduziert ist. Die einzelnen Komponenten des Webinterfaces werden im folgenden Abschnitt näher betrachtet.

4.3.5 Komponenten und Bedienung von BRIME

In Tabelle 4.25 sind die Komponenten der Webseite, ihre Lokalisation sowie ihre Funktion eingetragen. In den danach folgenden Kapiteln werden diese Funktionen im Detail veranschaulicht.

Tabelle 4.25: Übersicht über die einzelnen Komponenten des Webinterfaces BRIME sowie ihrer jeweiligen Funktion und ihrer Lokalisation im Frameset der Webseite

Komponente	Funktion	Lokalisation
Network Viewer	Anzeige der Stoffwechselkarte	Mittlere Frame
Navigation Control Panel	Navigation innerhalb der Karte	Rechter Frame
Übersichtskarte	Übersicht sowie direkte Navigation	Rechter Frame
Navigationsbaum	Auswahl organismusspezifischer Netzwerke, Stoffwechselweg orientierte Navigation	Linker Frame
Interaktive Suche	Interaktive Suche für Substanzen, Metaboliten und Stoffwechselwege	Linker Frame
Suche Reaktionen	Suchfunktion für Reaktionen und Teilreaktionen	Linker Frame
Sammelbox	Sammelbox für die Ausführung einer Suchanfrage	Linker Frame
Menüleiste	Beherbergt die Menüpunkte „Upload“, „Data Mapping“, „Examples“, „Download und Settings“.	Oberer Frame

4.3.5.1 Network Viewer

Der „Network Viewer“ im mittleren Frame ist die zentrale Komponente des Webinterfaces und zeigt die Stoffwechselkarte. Diese wird dynamisch durch einen serverseitigen Dämon erzeugt (siehe Kapitel 4.2). Er verarbeitet die Anfrage des Clients über einen Apache-Webserver, wobei PHP-Skripte mit dem C++-Programm über eine Socket-Schnittstelle kommunizieren. Der Server gibt das gerenderte Bild als PNG-Format zurück. Gleichzeitig wird eine ImageMap (deutsch: die verweissensitive Grafik) für das gerenderte Bild berechnet und zu dem Client übermittelt. Somit sind die Knoten- und Stoffwechselweg-Positionen bekannt und für weitergehende Informationsabfragen zugänglich. Hierfür wurde die OverLIB (siehe Kapitel 3.7.8) eingebunden, welche knotenspezifische Daten zu den darunter befindlichen Substanzen und Enzymen repräsentiert. Diese umfassen

weitergehende Verlinkungen, eine Molekülstruktur für Substanzen sowie die Möglichkeit, die Knoten zu Suchanfragen zu addieren bzw. die Knotennachbarn darzustellen und im Auswahlbildschirm zu fokussieren und einzupassen. Auch Verlinkungen zu anderen Datenbanken werden angezeigt. Abbildung 4.90 gibt ein Beispiel für eine Substanz-spezifische Information, Abbildung 4.91 für eine Enzym-spezifische Information.

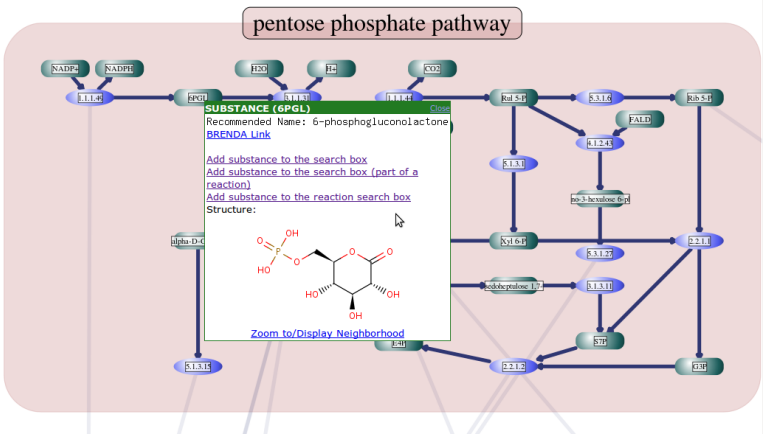

Abbildung 4.90: Beispiel für ein Substanz-spezifisches Informationsfenster für GPGL. In dem Informationsfenster werden der Gebrauchsname sowie eine BRENDA-Verlinkung angezeigt. Die Substanz kann zu der interaktiven Sammelbox hinzugefügt werden. Die Molekülstruktur wird als 2D Bild visualisiert und man kann in die Nachbarschaft wechseln.

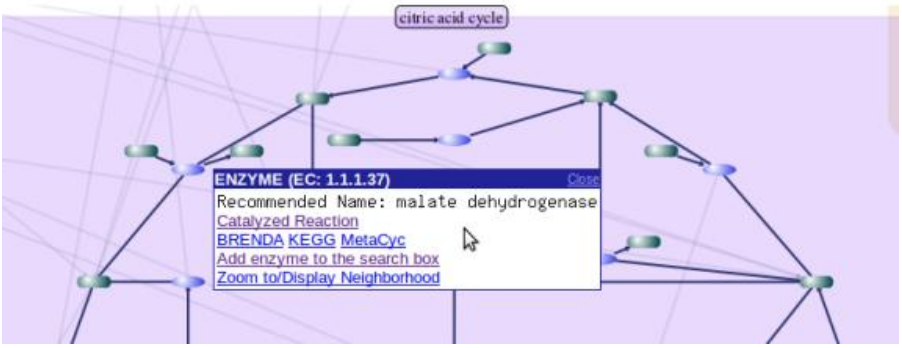

Abbildung 4.91: Beispiel für ein Enzym-spezifisches Informationsfenster für das Enzym Malat Dehydrogenase. In dem PopUp-Fenster werden der Gebrauchsname sowie Verlinkungen zu der katalysierten Reaktion, BRENDA, KEGG und MetaCyc angezeigt. Das Enzym kann zu der Sammelbox hinzugefügt werden. Man kann auch hier in die Nachbarschaft des Enzyms wechseln.

Für eine intuitive Bedienung wurde, mit Hilfe von „event-Handling“ (deutsch: die Ereignis-Behandlung) durch JavaScript, eine Zieh-Navigation per „Drag and Drop“ sowie eine Zoom-Funktion mittels Mausrad ermöglicht. Des Weiteren können die auf der Tastatur befindlichen Pfeiltasten für die Bewegung innerhalb der Karte benutzt werden. Alternative hierzu ist eine Navigation durch das im Kapitel 4.3.5.2 beschriebene „Navigation Control Panel“ möglich. Wie bereits erwähnt, kann der Benutzer eine individuelle Größenanpassung des gewünschten Kartenausschnitts vornehmen, in dem er die Größe der Frames im Frameset variiert.

4.3.5.2 Navigation Control Panel

Abbildung 4.92: Das dargestellte „Navigation Control Panel“ bietet 8 verschiedene Navigationsrichtungen sowie die Möglichkeit, die Zoomstufe schrittweise zu verändern oder direkt vorzugeben.

Das „Navigation Control Panel“ (Abbildung 4.92) befindet sich im rechten Frame der Webseite von BRIME. Es bietet die Möglichkeit, unabhängig von der implementierten Zieh-Navigation und Zoom-Funktion, den Kartenausschnitt im mittleren Frame flexibel zu manipulieren. Es werden acht verschiedene Navigationsrichtungen angeboten. Neben der Navigation in der Ebene kann ebenfalls schrittweise herein- und heraus-gezoomt werden sowie der Zoomfaktor direkt angegeben werden.

4.3.5.3 Übersichtskarte

Abbildung 4.93: Die obere Grafik zeigt die Übersichtskarte. Hier kann der Benutzer durch Anklicken direkt zu dem jeweiligen Punkt in der Stoffwechselkarte springen.

Im rechten Frame der Webseite befindet sich zusätzlich eine miniaturisierte Übersichtskarte (Abbildung 4.93), um dem Benutzer eine weitere Navigationshilfe zu offerieren. Es wird hierbei eine verkleinerte Version der generischen Stoffwechselkarte dargestellt. Durch Klicken in die Karte wird die relative Kartenposition an den C++-Server übertragen, welcher diesen Punkt auf die Stoffwechselkarte, visualisiert durch den „Network Viewer“, transformiert und automatisch zu diesem Kartenpunkt springt.

4.3.5.4 Interaktive Suche

Im linken Frame befindet sich die interaktive Suche (Abbildung 4.94). Hier gibt es die Möglichkeit, Substanzen, Enzyme sowie Reaktionen zu suchen. Die Suche wird hierbei interaktiv durch ein „LiveSearch“ (siehe Kapitel 3.7.7) überprüft und mit den Einträgen in der Datenbank verglichen, wobei Suchtreffer dynamisch angezeigt werden. Hierbei werden auch Synonyme mit in die Suche einbezogen. Die Suchergebnisse werden in einem Navigationsbaum dargestellt und der Benutzer kann gefundene Anfragen auszuwählen. Diese werden in der metabolischen Karte fokussiert.

BRIME bietet zusätzlich die Möglichkeit, eine Reaktionssuche durchzuführen. Die Verarbeitung der Anfrage erfolgt hierbei, wie in Kapitel 3.9.6 beschrieben. Die Ergebnisse werden ebenfalls im Navigationsbaum gelistet.

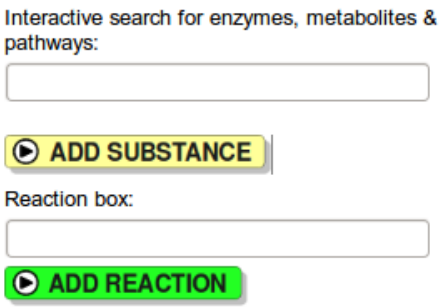

Abbildung 4.94: BRIME bietet eine interaktive Suche für Substanzen, Enzyme und Stoffwechselwege. Die Suche von Reaktionen ist ebenfalls möglich. Die Suchanfrage wird hierbei von dem in Kapitel 3.9.6 vorgestellten Tool verarbeitet.

4.3.5.5 Sammelbox

Die Sammelbox (siehe Abbildung 4.95) ist der zentrale Bereich, um Suchanfragen für die finale Suchausführung zu speichern. So kann der Benutzer neben der interaktiven Suche auch durch Einfügen eines eigenen Textes in diese Textbox die Suchanfrage nach seinen Wünschen gestalten. Die Erstellung einer solchen Suchanfrage wird als Anwendungsbeispiel in Kapitel 4.3.7.4 im Detail gezeigt.

Abbildung 4.95: Die Sammelbox bietet textbasiert Manipulationsmöglichkeiten der benutzerorientierten Suchanfrage. Durch Anklicken von „Search“ wird diese dem BRIME-Server übermittelt.

4.3.5.6 Navigationsbaum

Visualisierung organismusspezifischer Karten

Der Navigationsbaum dient als Benutzerinterface, um komfortabel zwischen der generischen Stoffwechselkarte, organismusspezifischen Karten, benutzerspezifischen Karten sowie den gestellten Suchanfragen navigieren zu können (siehe Abbildung 4.96). Die Baumknoten lassen sich hierbei aufklappen und weitere Navigationsmöglichkeiten, wie z.B. die Stoffwechselweg-orientierte Navigation, werden angeboten.

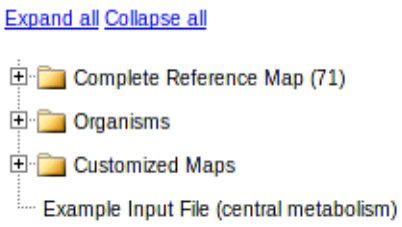

Abbildung 4.96: Die obersten Knoten des Navigationsbaums ermöglichen die Auswahl zwischen der generischen Stoffwechselkarte, organismusspezifischen Karten, benutzerspezifischen Karten sowie den gestellten Suchanfragen.

Der Navigationsbaum enthält drüber hinaus die in der Datenbank gespeicherten Organismen (Abbildung 4.97) und die dazugehörigen organismusspezifischen Teilnetzwerke. Es werden Annotationen der BRENDA sowie KEGG Datenbanken angeboten. Die Annotationsquelle wird in eckigen Klammern hinter dem Namen des Organismus angezeigt (Abbildung 4.97). Wählt man einen Organismus , so werden nur die Enzyme grafisch hervorgehoben, die in dem jeweiligen Organismus vorhanden sind. Durch die Navigation innerhalb des Baums können die jeweiligen Datensätze in der Karte visualisiert werden.

Abbildung 4.97: Beispiel einer Auswahl einer Organismus-spezifischen Karte: Das Netzwerk von *Escherichia coli* ist selektiert. Über einen Informationstext wird angezeigt, dass die Annotation aus der BRENDA-Datenbank stammt.

Visualisierung benutzerspezifischer Karten

Dem Benutzer von BRIME stehen drei Karten zur Verfügung. Diese können durch das Hochladen von Datensätzen, beschrieben in Kapitel 4.3.5.7, verändert werden. Diese benutzerspezifischen Karten befinden sich unter dem Knoten von „Customized Map“ des Navigationsbaumes. Die Gestaltung einer benutzerspezifischen Karte wird in Kapitel 4.3.7.5 anhand eines Anwendungsbeispiels näher erläutert.

Stoffwechselweg-orientierte Navigation

Sowohl der Baumknoten der generischen Gesamtstoffwechselkarte als auch die Knoten für die organismusspezifischen Annotationen bieten eine Anzahl von Stoffwechselwegen als Kindknoten. Durch manuelle Kuration wurden für die verschiedenen Organismen nur diejenigen Stoffwechselwege aussortiert, die in dem Organismus vorliegen können. Abbildung 4.98 zeigt exemplarisch die alphabetisch sortierte Liste der Stoffwechselwege von *Escherichia coli* bezüglich der BRENDA Genome/Enzym-Annotation. Eine

detaillierte Betrachtung findet sich in dem Anwendungsbeispiel „Stoffwechselweg-orientierte Navigation“ in Kapitel 4.3.7.3.

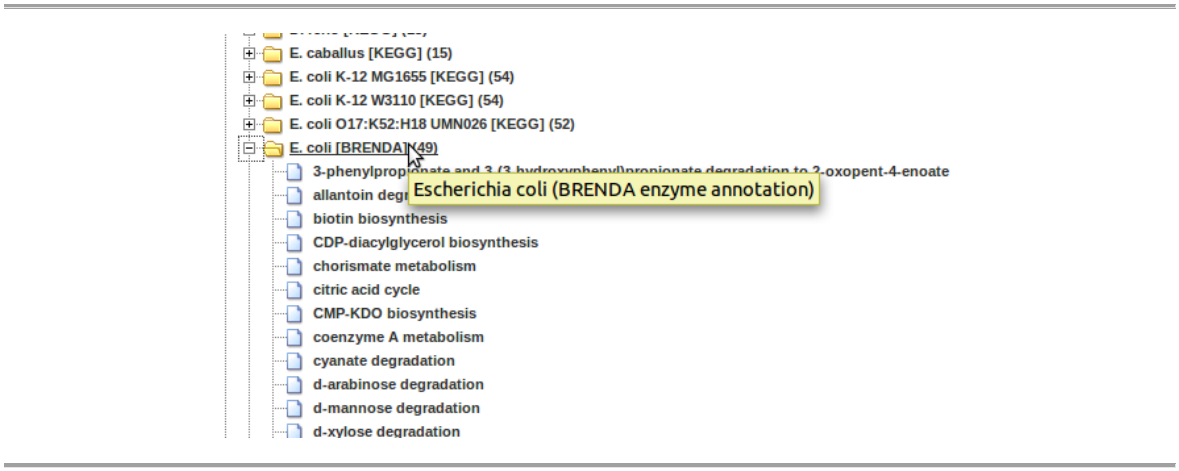

Abbildung 4.98: Beispiel einer Auswahl eines spezifischen Stoffwechselwegs. Jeder Organismus-Knoten beinhaltet eine Reihe ihm zugehöriger Stoffwechselwege. Durch Auswahl wird automatisch dieser Stoffwechselweg im „Network Viewer“ fokussiert.

Suchanfragen-orientierte Navigation

Die in Kapitel 4.3.5.5 beschriebene Sammelbox dient der Sammlung von Suchanfragen. Wird final eine aggregierte Suchanfrage ausgeführt, werden die gefundenen Treffer in dem Navigationsbaum gelistet. Der Benutzer kann Treffer im Suchbaum auswählen. Die zuletzt gewählte Karte ist hierbei mit dem Suchbaum gekoppelt und der Kartenausschnitt fokussiert auf die Auswahl im Baum. Durch Wechsel der Karten können Treffer innerhalb der generischen, den organismusspezifischen oder den benutzerspezifischen Karten durch den „Network Viewer“ angezeigt werden. Im Anwendungsbeispiel „Erstellen von Suchanfragen“ in Kapitel 4.3.7.4 wird dieses exemplarisch veranschaulicht.

4.3.5.7 Menüleiste

Die Menüleiste bietet dem Nutzer die Möglichkeit, Daten zum Server zu laden bzw. Datensätze herunterzuladen. Außerdem befinden sich diverse Einstellmöglichkeiten in den Reitern „Data Mapping“ sowie „Settings“. Über die Menüleiste kann der Benutzer eine umfassende Hilfeseite aufrufen. Die wichtigsten Elemente des Menüs werden im Folgenden beschrieben.

Menüpunkt „Upload“

Unter dem Menüpunkt „Upload“ können Datensätze im CSV-Format zu dem BRIME-Webserver geladen werden. Hier können Daten für Metabolite, Enzyme oder Flussdaten übertragen werden. Als Referenz für die Codierung dient die generische Gesamtstoffwechselkarte. Die jeweiligen Datensätze werden zu der ausgewählten benutzerspezifischen Karte geladen und auf dieser abbgebildet. Es stehen hierbei drei Karten zur Verfügung. Der Benutzer kann zwischen den Möglichkeiten wählen, die Daten zu addieren, subtrahieren oder neu zu setzen. In Abbildung 4.99 ist das modale Fenster gezeigt, welches sich öffnet, wenn man ein Metabolit-Datensatz abbilden möchte.

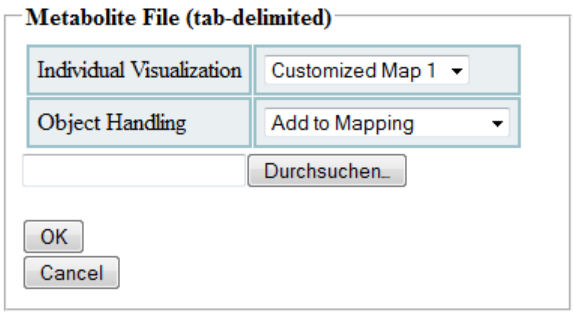

Abbildung 4.99: Beim „Upload“ öffnet sich ein modales Fenster, in dem die gewünschte benutzerspezifische Karte sowie die Art des Datenhandlings bestimmt werden kann. In dem Fenster wird exemplarisch der Upload eines Metabolit-Datensatzes gezeigt.

Menüpunkt „Data Mapping“

Sofern ein Datensatz geladen wurde, kann das Codierungsverhalten angepasst werden. Hierbei stehen dem Nutzer die Möglichkeiten zur Verfügung, das grafische Darstellungsverhalten der Knoten (siehe Abbildung 4.100) und der Kanten (siehe Abbildung 4.101) zu verändern.

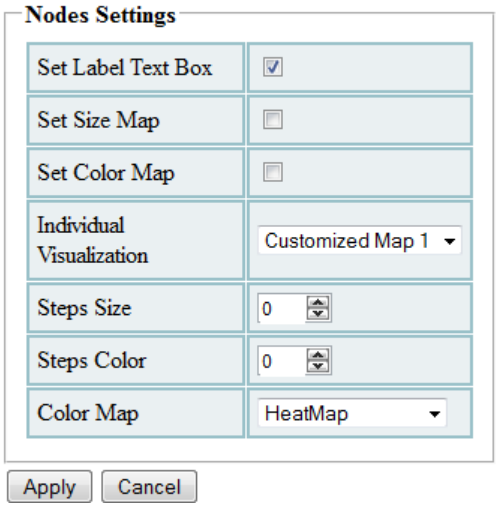

Abbildung 4.100: Durch Anklicken des Menüpunktes „Data Mapping Nodes“ öffnet sich das modale Fenster „Nodes Settings“. Es können diverse Parameter angepasst werden, die die Visualisierung der Daten bezüglich der Knoten verändert.

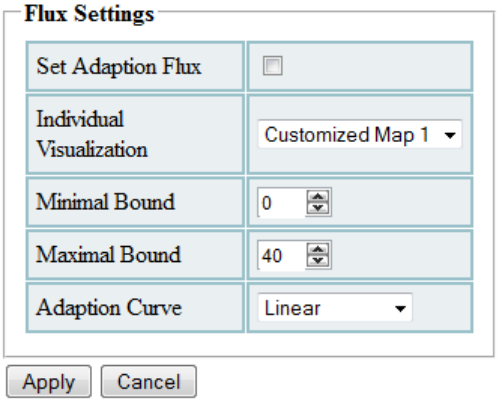

Abbildung 4.101: Das modale Fenster „Flux Settings“ öffnet sich durch Anklicken des Menüpunktes „Data Mapping Flux“. Es können die Codierungs-Parameter vorgegeben werden, die Einfluss auf die grafische Darstellung der Kanten haben.

Menüpunkt „Examples“

Sofern der Benutzer Beispieldateien laden möchte, um sich beispielsweise mit der Datenstruktur der Datei vertraut zu machen oder die Codierungsmöglichkeiten von BRIME zu testen, bietet der Menüpunkt „Examples“ drei Beispielldateien. Die Dateien sind im

CSV-Format hinterlegt und es ist jeweils eine Datei für eine Substanz-basierte, eine Enzym-basierte und ein Fluss-basierte Codierung vorhanden.

Menüpunkt „Download“

Wurden in BRIME Datensätze durch einen Benutzer geladen, so wird der Abgleich der Daten mit der generischen Karte gespeichert. Unter dem Menüpunkt „Download“ wird dem Anwender dieser Abbgleich zum Download angeboten. BRIME verwaltet hierbei sowohl gefundene als auch nicht gefundene Einträge. In dem Anwendungsbeispiel „Analyse von –ome Daten“ (Kapitel 4.3.7.5) wird dies exemplarisch gezeigt.

Menüpunkt „Settings“

Unter diesem Menüpunkt findet der Benutzer von BRIME alle zentralen Konfigurationsmöglichkeiten der Webseite. Hierzu gehören die Parameter des „Network Viewers“ (Kapitel 4.3.5.1), des Navigationsbaums (Kapitel 4.3.5.6) und die der Suche (Kapitel 4.3.5.4). Unter „Settings View“ werden Einstellungen für den „Network Viewer“ bearbeitet. Abbildung 4.102 zeigt das Dialogfenster. Möchte der Benutzer automatisch Knoten durch Darüberfahren mit der Maus selektieren, so kann er die Checkbox neben „Automatic Selection“ aktivieren. Die selektierten Knoten werden innerhalb der Sammelbox (Kapitel 4.3.5.5) gelistet. Das „Drag and Drop“ Verhalten kann über die dafür vorgesehene Checkbox an- bzw. ausgeschaltet werden. Der Benutzer kann wählen, ob Informationsfenster, erzeugt aus der OverLIB (Kapitel 3.7.8), eingeblendet werden sollen oder nicht.

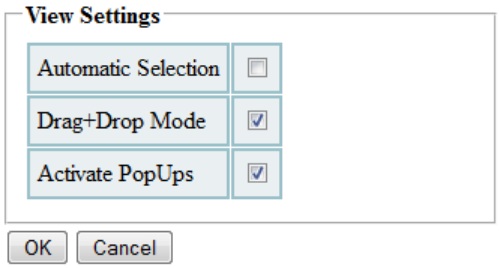

Abbildung 4.102: In dem modalen Fenster „View Settings“ werden Parameter für den „Network Viewer“ angepasst Hier kann das Selektionsverhalten, das „Drag and Drop“-Verhalten sowie das Anzeigen von Informationsfenstern (PopUps) aktiviert oder deaktiviert werden.

Wird der Menüpunkt „Settings Tree“ gewählt, öffnet sich das Fenster aus Abbildung 4.103. Der Parameter „Focus on Neighbourhood“ gibt hierbei an, ob bei Auswahl eines Netzwerkknotens automatisch auf dessen Nachbarschaft fokussiert werden soll oder nicht. Dieses Verhalten kann hilfreich sein, um die zugehörige katalysierte Reaktion eines Enzyms innerhalb des Netzwerks zu lokalisieren. Der Parameter „Threshold“ gibt an, wie im Navigationsbaum Stoffwechselwege gelistet werden sollen. Es handelt sich um die Vorgabe eines prozentualen Schwellwertes, der definiert, wie viel Prozent der Enzyme im Vergleich zu dem Stoffwechselweg in der generischen Gesamtstoffwechselkarte mindestens vorhanden sein müssen, damit er innerhalb des Navigationsbaums aufgeführt wird.

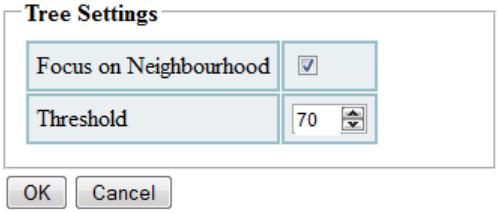

Abbildung 4.103: Innerhalb des modalen Fensters „Tree Settings“ können Parameter für den Navigationsbaum übernommen werden. Bei Auswahl eines Enzyms kann automatisch auf dessen Nachbarknoten fokussiert werden. („Focus on Neighbourhood“). Außerdem kann der Schwellenwert angegeben werden, für den Stoffwechselwege für Organismen aussortiert werden, sofern sie nicht mindestens die prozentuale Anzahl von Enzymen im Vergleich zu der generischen Gesamtstoffwechselkarte aufweisen.

Einstellungen, die Sucheigenschaften betreffen, können unter „Settings Search“ verändert werden (siehe Abbildung 4.104). Es kann bestimmt werden, welche biochemisch relavanten Objekte gesucht werden sollen. Es kann nach Enzymen, Substanzen, Stoffwechselwegen und Reaktionen gesucht werden. Der Baum, der die Suchanfrage wiedergibt, wird unter Berücksichtigung der Einstellung entsprechend aufgebaut.

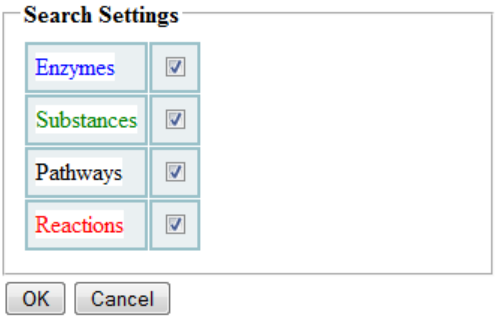

Abbildung 4.104: Das modale Fenster „Search Settings“ bietet Einstellmöglichkeiten für die Suche. Es kann definiert werden, ob nach Enzymen, Substanzen, Stoffwechselwegen oder Reaktionen gesucht werden soll.

Unter dem Menüpunkt „Help“ findet sich eine detallierte Hilfeseite. Durch die hier beschriebenen Anwendungsfälle, sollen die Möglichkeiten von BRIME aufzeigt werden. Unter dem Menüpunkt „Contact“ sind die Kontakte der Entwickler gelistet und im Menüpunkt „Links“ werden Verlinkungen zu metabolischen Datenbanken aufgeführt.

4.3.6 Grafische Anpassung der Netzwerkobjekte

Die grafische Gestaltungsmöglichkeit eines metabolischen Netzwerks bei BRIME ist gegenüber dem Pathway-Editor MapOmnia eingeschränkt. BRIME bietet keine direkte Möglichkeit die Netzwerkobjekte (Knoten, Kanten und Gruppen) zu manipulieren. Dennoch kann das Aussehen der Objekte über Laden von Daten und Suchanfragen verändert werden. Die grafischen Anpassungsmöglichkeiten der Netzwerkobjekte werden in folgenden Unterabschnitten aufgezeigt.

4.3.6.1 Grafische Anpassung der Knoten

Werden dem Server Datensätze übermittelt, die auf der Referenzkarte abgebildet werden sollen, werden gefundene Knoten eingeblendet und markiert. Nicht gefundene Knoten werden hingegen transparent dargestellt. Wurde ein Datensatz für eine Größencodierung geladen, werden zusätzlich noch die Größen der gefundenen Knoten mit den dazugehörigen Werten transformiert (vergleiche Kapitel 2.1.5.4). Analoges erfolgt für eine Farbcodierung, wobei hier die Farbe der Knoten angepasst wird (vergleiche Kapitel

2.1.5.3). Das Darstellungsverhalten ist exemplarisch in Abbildung 4.105 dargestellt. In Kapitel 4.3.7.5 wird auf das Visualisieren von Daten im Detail eingegangen.

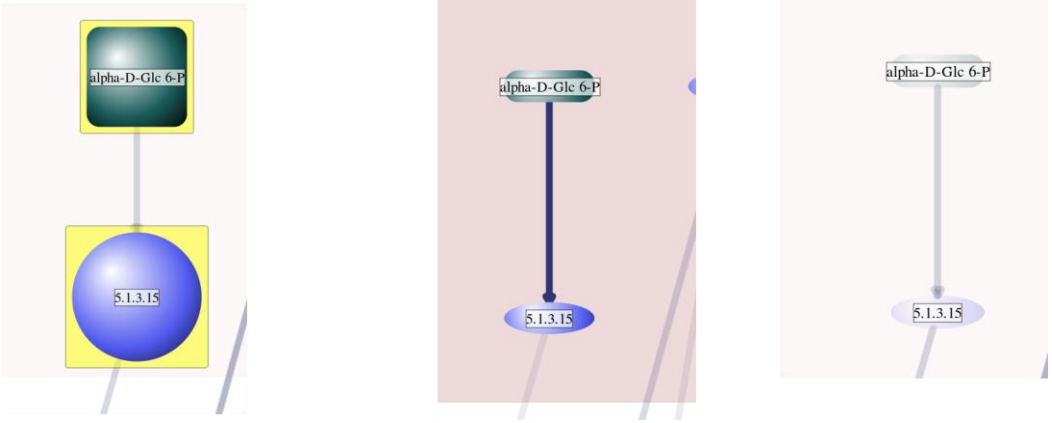

Abbildung 4.105: Beispielhaft wird ein Knoten des Typs Enzym und ein Knoten des Typs Metabolit dargestellt. Die mittlere Darstellung entspricht hierbei der Basisdarstellung der generischen Gesamtstoffwechselkarte. In der linken Darstellung wurden Metabolomdaten und Enzymdaten abgebildet, was zu einer Veränderung der Größe führt. In der rechten Darstellung werden die Knoten transparent gerendert, da sie für die Abbildung der Daten nicht gefunden wurden.

4.3.6.2 Grafische Anpassung der Kanten

Werden Fluss-basierte Daten geladen, so erfolgt eine Anpassung der Kanten. Hierbei wird neben der Strichstärke auch die Kantenrichtung der Flussrichtung gemäß angepasst. In Abbildung 4.105 wird dieses Darstellungsverhalten demonstriert und im Anwendungsbeispiel in Kapitel 4.3.7.5 für reale Daten gezeigt.

4.3.6.3 Grafische Anpassung der Gruppen

Die grafische Anpassungsmöglichkeit der Gruppen in BRIME ist sehr eingeschränkt und wie auch für die Anpassung anderer Netzwerkobjekte nur indirekt durch das Laden von Daten zu steuern. Gruppen können hierbei ein- und ausgeblendet dargestellt werden, wie in Abbildung 4.105 exemplarisch gezeigt wird. Im Anwendungsbeispiel in Kapitel 4.3.7.5 wird auf die Anpassung detailliert eingegangen.

4.3.7 Anwendungsbeispiele von BRIME

Um einen tieferen Einblick in BRIME zu gewähren, werden in den folgenden Kapiteln mehrere Anwendungsbeispiele beschrieben. Zunächst wird in Kapitel 4.3.7.1 die Ansicht der generischen Stoffwechselkarte erläutert. Darauf aufbauend wird in Kapitel 4.3.7.2 die Ansicht organismusspezifischer Stoffwechselkarten erörtert. Kapitel 4.3.7.3 behandelt die Stoffwechselweg-orientierte Navigation und die Erstellung von Suchanfragen wird in Kapitel 4.3.7.4 gezeigt. Abschließend wird in Kapitel 4.3.7.5 eine Analyse von „–ome“-Daten durchgeführt.

4.3.7.1 Ansicht der generischern Stoffwechselkarte

Wird die Webapplikation BRIME geöffnet, zeigt diese zunächst die globale generische Stoffwechselkarte (siehe Abbildung 4.106). Diese globale Referenzkarte kann hierbei als Zusammenfassung der zentralen biochemischen Reaktionen betrachtet werden, welche in allen lebenden Organismen vorkommen. Um die Karte in einer verständlichen und übersichtlichen Weise zu gestalten, wurde allerdings darauf verzichtet, biochemische Prozesse, welche DNA und RNA betreffen (wie die DNA-Replikation und -Transkription sowie der sekundäre Metabolismus der Pflanzen) zu integrieren. Dieses globale Netzwerk enthält über 1300 verschiedene Enzyme und über 1400 verschiedene chemische Substanzen, welche als Edukte und Produkte der Enzyme fungieren. Die Karte repräsentiert hierbei über 1600 biochemische Reaktionen, die zu über 120 wohldefinierten Stoffwechselwegen zugeordnet sind.

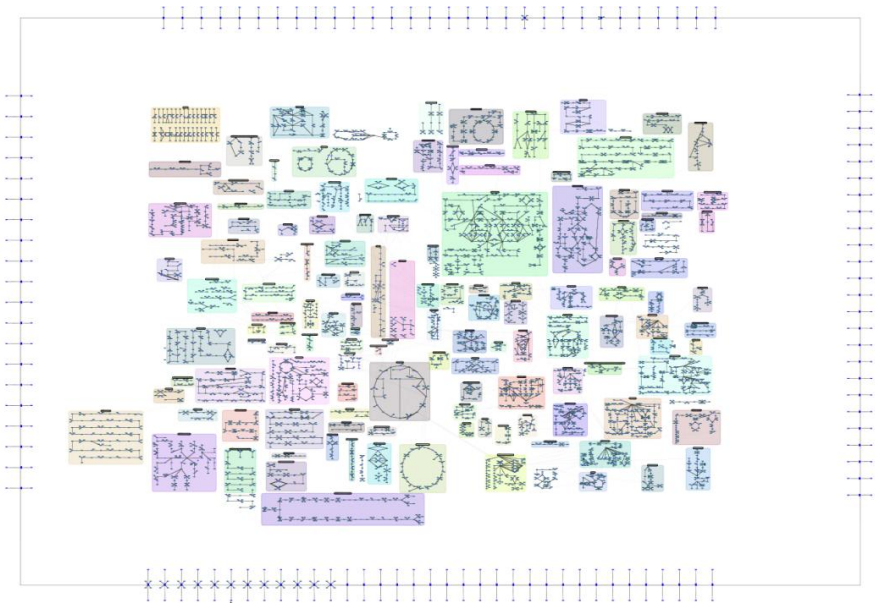

Abbildung 4.106: Die generische Stoffwechselkarte dargestellt in BRIME.

Innerhalb dieser Karte ist es durch die beschriebenen Navigationsmöglichkeiten (siehe Kapitel 4.3.5.1, und 4.3.5.2 sowie 4.3.5.3) einfach möglich zu navigieren und die Karte zu erkunden. In Abbildung 4.107 ist exemplarisch ein Ausschnitt der Karte gezeigt, welcher vergößert wurde.

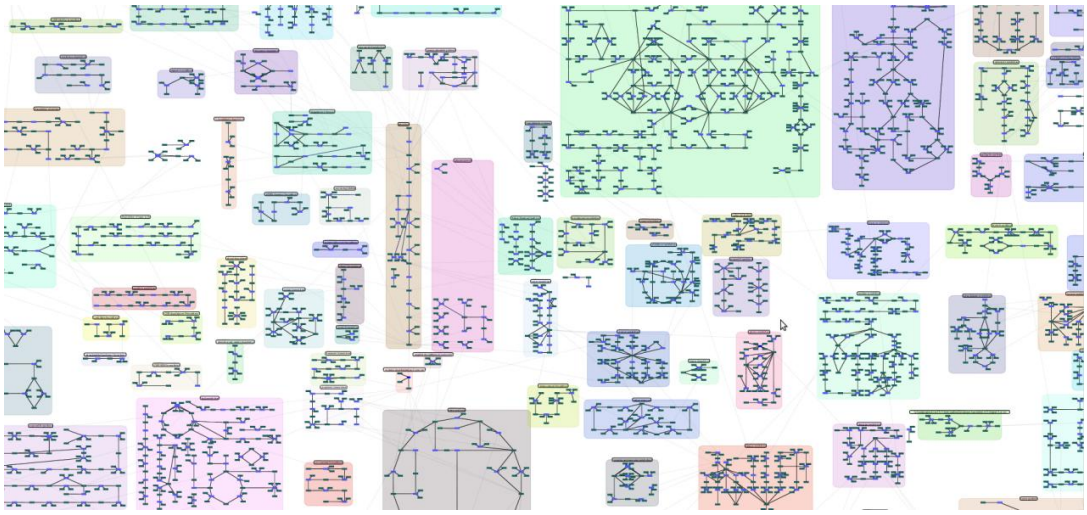

Abbildung 4.107: In BRIME können Stoffwechselkarten erkundet werden. Die Abbildung zeigt einen vergrößerten Bereich, auf den der Benutzer herangezoomt hat.

4.3.7.2 Ansicht organismusspezifischer Stoffwechselkarten

BRIME bietet dem Nutzer die Möglichkeit, zwischen verschiedenen organismusspezifischen Stoffwechselkarten zu wählen und diese zu visualisieren. Dies wird im Folgenden anhand eines Vergleiches zwischen der Gesamtstoffwechselkarte und dem Metabolismus von *Escherichia coli* bezüglich der BRENDA-Annotation aufgezeigt.
In Abbildung 4.108 wird ein Ausschnitt der generischen Gesamtstoffwechselkarte gezeigt. Der Fokus ist auf den zentralen Metabolismus gelegt.

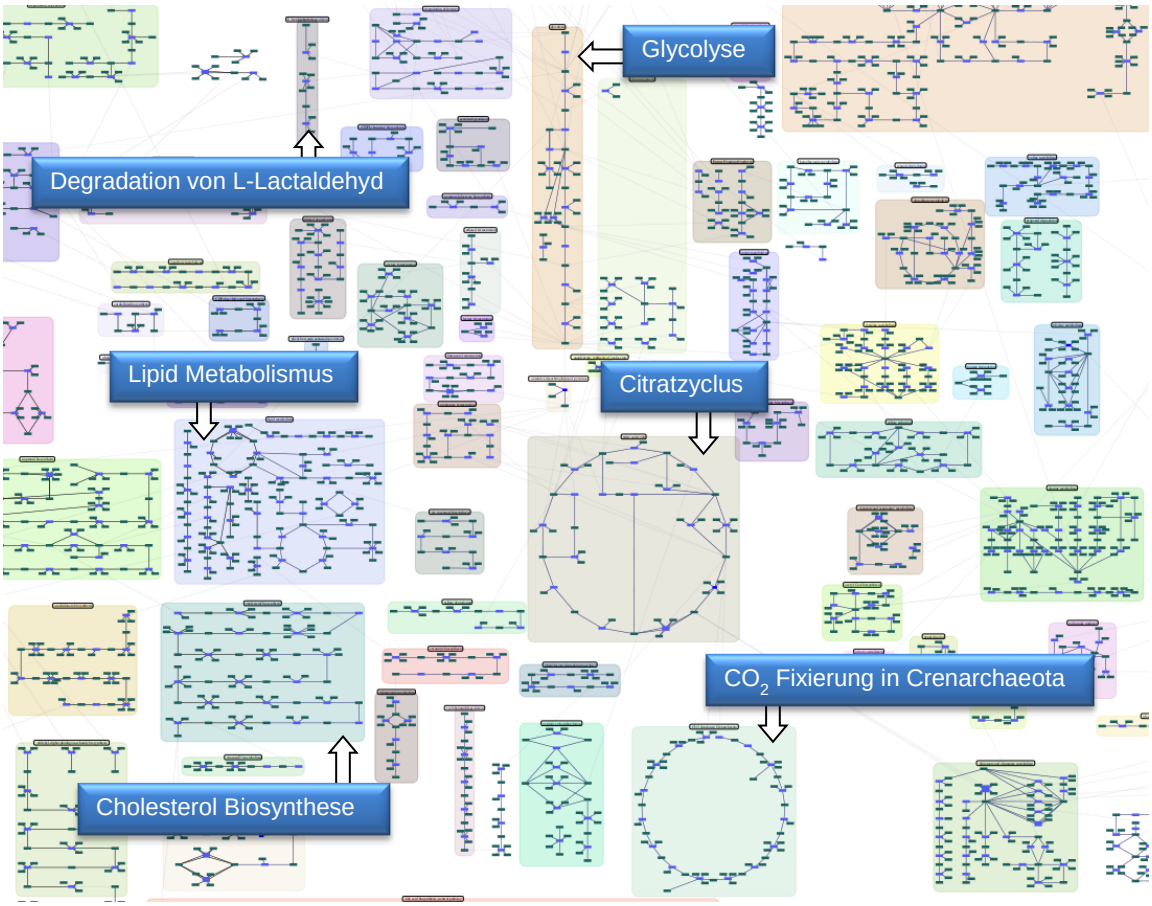

Abbildung 4.108: Die generische Stoffwechselkarte von BRENDA dargestellt in BRIME mit dem Fokus auf den zentralen Metabolismus.

Hierbei werden sechs metabolische Stoffwechselwege herausgestellt. Zunächst diejenigen, die üblicherweise in den meisten Organismen vorkommen. Hierzu gehören die Stoffwechselwege des zentralen Metabolismus (Glykolyse und der Citratzyklus) und der

Lipid-Metabolismus sowie die Degradation von L-Lactaldehyd. Zusätzlich sind Stoffwechselwege markiert, welche charakteristisch für verschiedene taxonomische Gruppen sind, wie die Cholesterol-Biosynthese (Mammalia) und die CO_2 Fixierung in Crenarchaeota (Archaea).

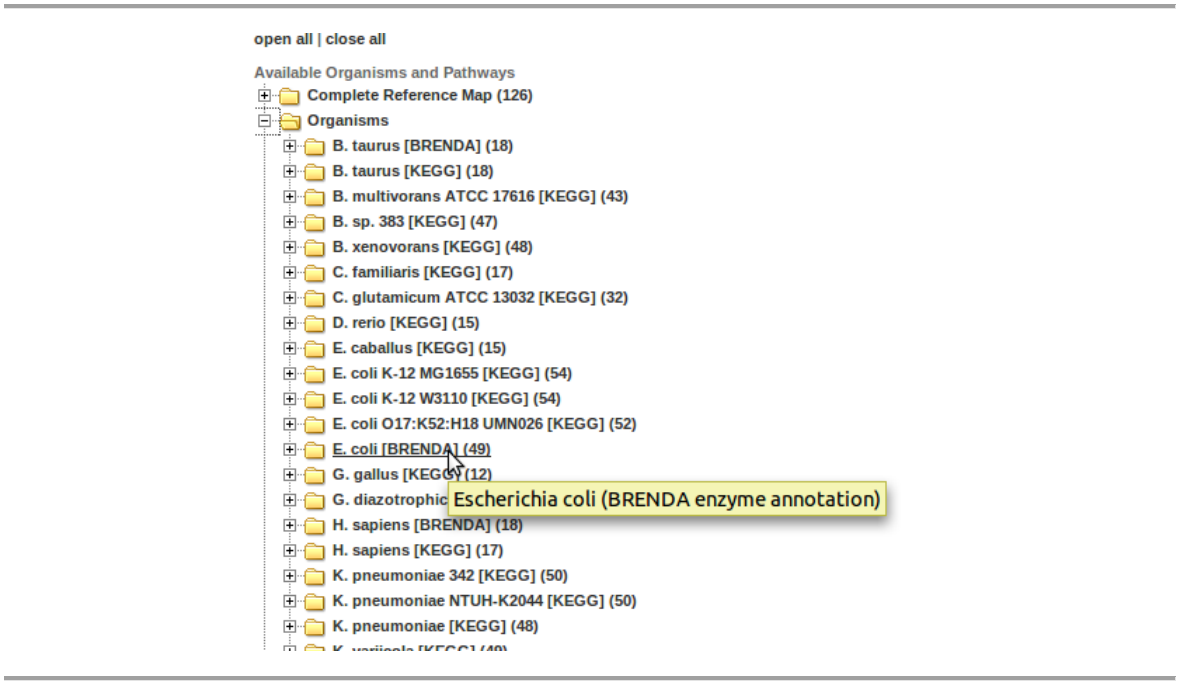

Abbildung 4.109: Ausschnitt aus dem Navigationsbaum von BRIME bei der Auswahl eines organismusspezifischen Teilnetzwerkes.

Wählt der Nutzer im Navigationsbaum unter dem Baumknoten „Organism“ das organismusspezifische Netzwerk für *Escherichia coli* (Abbildung 4.109), dessen Genome/Enzyme Annotation aus der BRENDA-Datenbank erhalten wurde, so zeigt sich die in Abbildung 4.110 dargestellte Stoffwechselkarte.

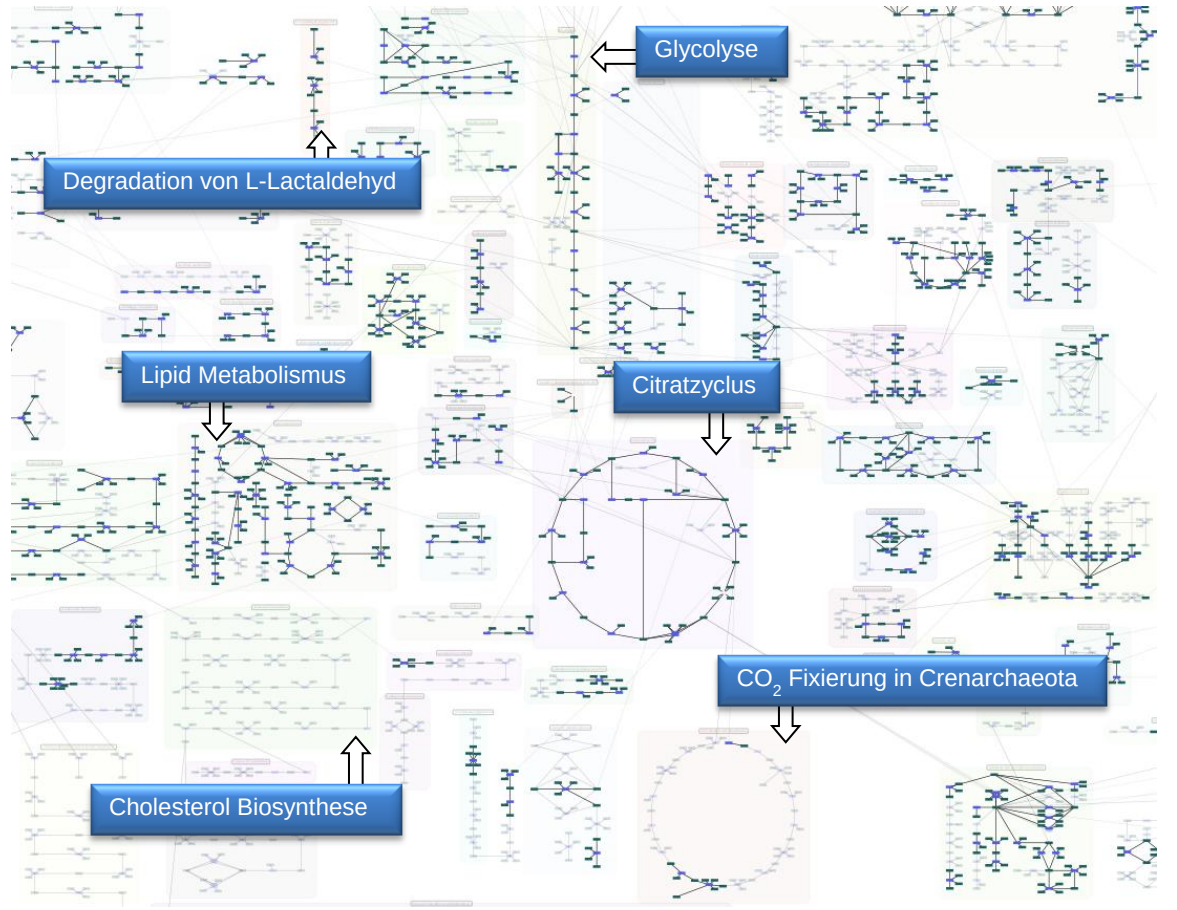

Abbildung 4.110: Der gleiche Kartenausschnitt wie aus Abbildung 4.108 mit Fokus auf den zentralen Stoffwechsel wird im „Network Viewer“ für die Selektion von Escherichia coli angezeigt.

Es werden bei der Ansicht diejenigen Enzyme eingeblendet und markiert, welche in dem Organismus vorkommen. Diejenigen, die nicht vorkommen, sind transparent dargestellt. Dies ermöglicht einen direkten visuellen Bezug zu der Gesamtstoffwechselkarte. So zeigt sich im Beispiel *Escherichia coli*, dass viele Enzyme von Stoffwechselwegen, wie die des zentralen Metabolismus, des Lipid Metabolismus sowie die Degradation von L-Lactaldehyd, die gewöhnlich in den meisten Organismen vorkommen, eingeblendet sind. Wie erwartet, werden Enzyme, die keine Rolle in Bakterien spielen, wie die Cholesterol-Biosynthese (Mammalia) und die CO_2 Fixierung in Crenarchaeota (Archaea) transparent dargestellt. Stoffwechselwege selbst werden farbig unterlegt hervorgehoben,

sofern 100% der in dem jeweiligen Stoffwechselweg vorliegenden Enzyme auch in dem Organismus vorhanden sind.

Innerhalb des Navigationsbaumes werden unter jedem Baumknoten der Organismen nur diejenigen Stoffwechselwege abgelegt, die zu dem Organismus gehören. So finden sich die beiden erwähnten Stoffwechselwege, die Cholesterol Biosynthese und die CO_2 Fixierung, nicht in dem Unterbaum von *Escherichia coli* (vergleiche Abbildung 4.111).

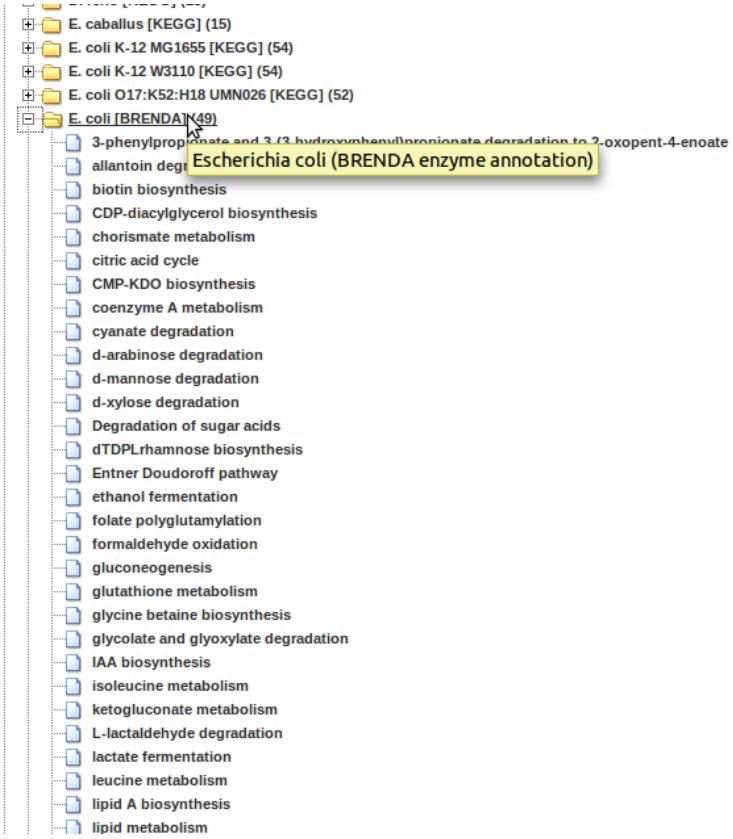

Abbildung 4.111: Ausschnitt aus dem Navigationsbaum von BRIME. Der Baumknoten für Teilnetzwerke beinhaltet als Kindknoten die dazugehörigen Stoffwechselwege, die relevant für den Organismus sind.

4.3.7.3 Stoffwechselweg-orientierte Navigation

Dem Benutzer wird von BRIME eine Stoffwechselweg-orientierte Navigation angeboten. Hierbei spielt es keine Rolle, ob der Benutzer die Referenzkarte oder eine organismusspezifische Ansicht gewählt hat. In Abbildung 4.112 ist wieder die Referenzkarte dargestellt, wobei exemplarisch Stoffwechselwege markiert sind.

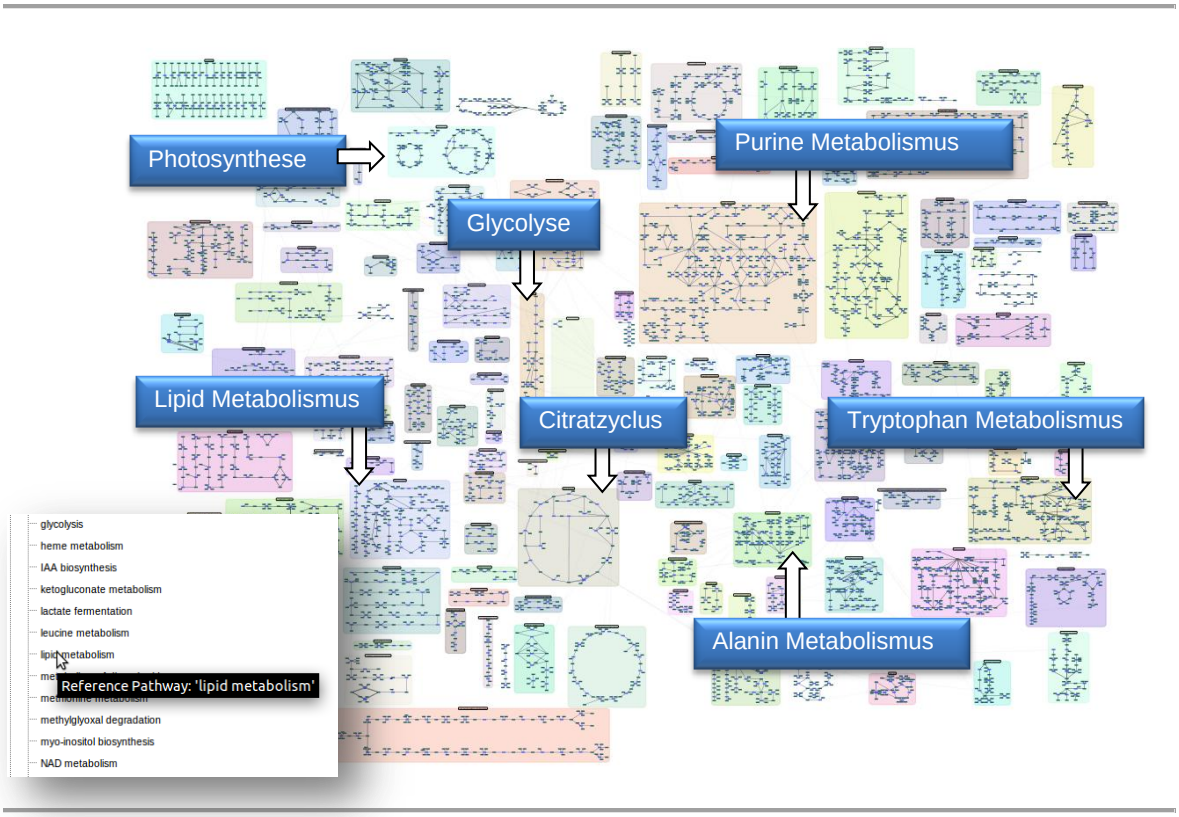

Abbildung 4.112: Auswahl eines spezifischen Stoffwechselwegs: Jeder Organismus-Knoten beinhaltet eine Reihe ihm zugehöriger Stoffwechselwege. Durch Auswahl wird automatisch dieser Stoffwechselweg im „Network Viewer“ fokussiert.

Für die Gesamtstoffwechselkarte sind über 120 verschiedene Stoffwechselwege verfügbar. Die genaue Anzahl der Stoffwechselwege steht in der runden Klammer hinter dem jeweiligen Organismus bzw. der Referenzkarte. Die Auswahl eines Stoffwechselwegs kann durch einfaches Klicken auf den dazugehörigen Baumknoten erfolgen. Automatisch springt die Karte zu dem Stoffwechselweg und passt die Größe des Kartenausschnitts an (siehe Abbildung 4.113).

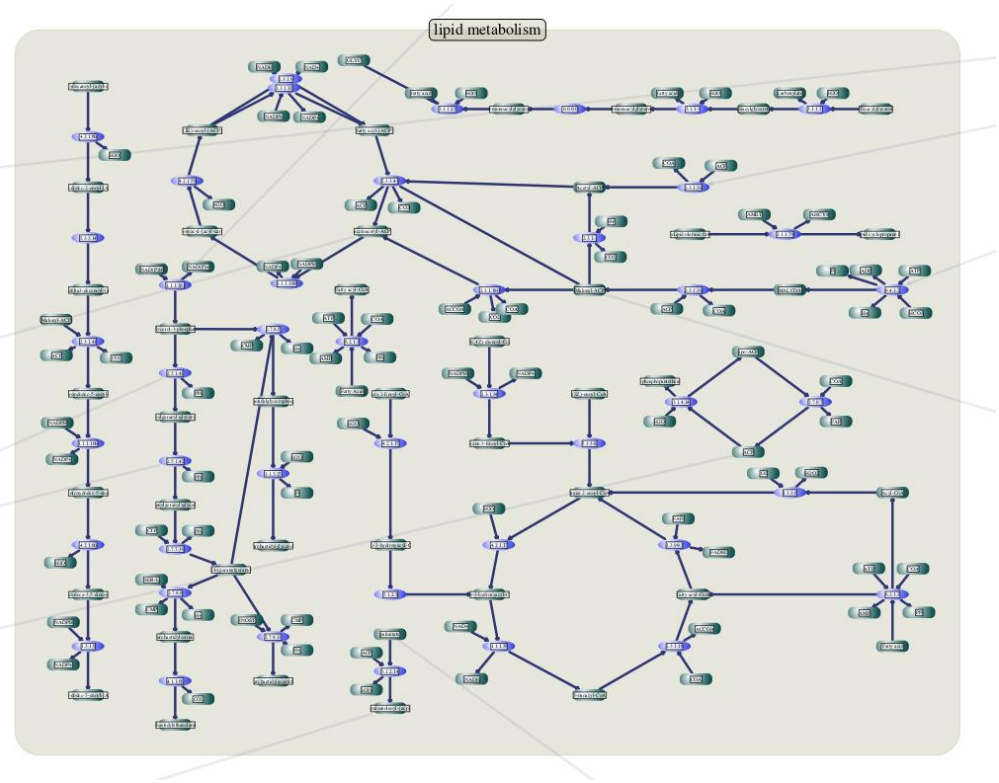

Abbildung 4.113**:** Durch Auswahl eines Stoffwechselweges wird automatisch auf diesen fokussiert. In diesem Beispiel wird der Lipid Metabolismus angezeigt.

Analog zu dem obigen Beispiel ist es möglich, stoffwechselwegbasiert innerhalb der Karte für die vorhandenen Organismen zu navigieren. Allerdings finden sich im Baum nur diejenigen Stoffwechselwege als Kindknoten, die für den jeweiligen Organismus relevant sind.

4.3.7.4 Erstellen von Suchanfragen

BRIME bietet eine umfassende Möglichkeit, Suchanfragen an das Netzwerk zu stellen. So kann enzymbasiert, substanzbasiert und ebenfalls reaktionsbasiert im Netzwerk gesucht werden. Dieses gilt auch für die Suche nach Stoffwechselwegen. In Abbildung 4.114 (linkes Bild) wird im Suchfeld für Enzyme, Metabolite und Stoffwechselwege beispielhaft der Anfang einer EC-Nummer eingegeben. Automatisch werden die Treffer als Autovervollständigung angeboten, welche in BRIME zu dieser Teil-EC-Nummer gefunden

werden. Es wird gleichzeitig die Anzahl der gefundenen Treffer angezeigt. Die Suchtreffer sind verlinkt, sodass sie bei Auswahl direkt zu der Sammelbox hinzugefügt werden.

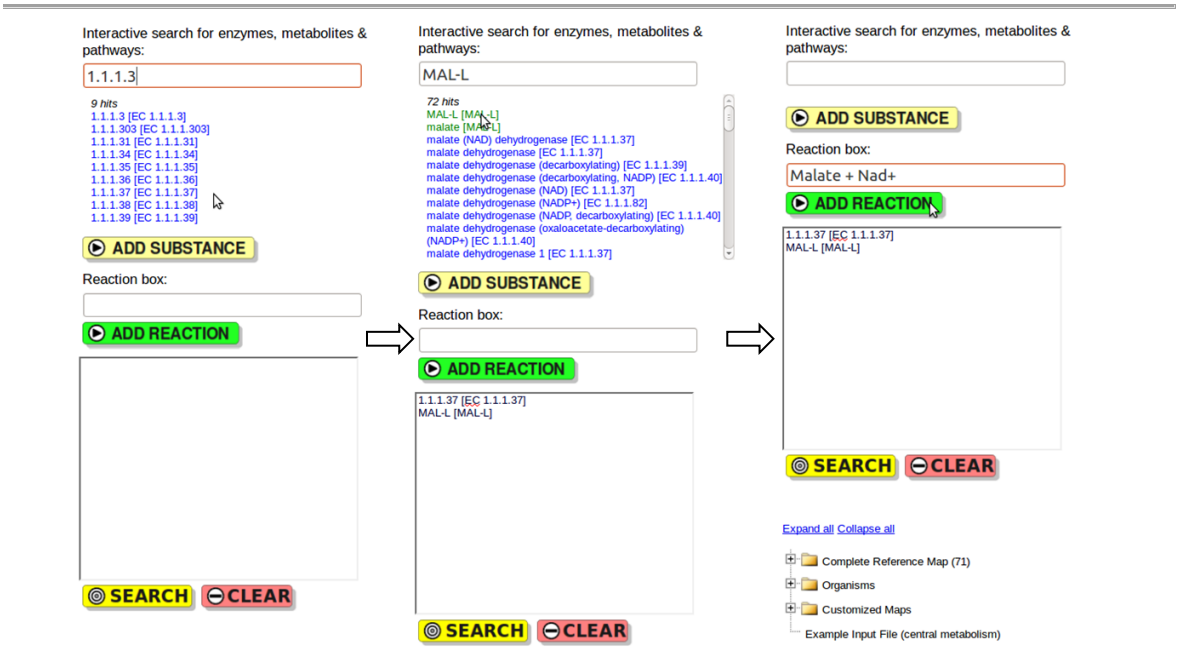

Abbildung 4.114: Erstellen einer Suchanfrage für ein Enzym welches mit 1.1.1.3 beginnt (linkes Bild) sowie nach dem Metaboliten MAL-L (mittleres Bild). Im rechten Bild wird die Teilreaktion „Malate + NAD+“ zu der Sammelbox hinzugefügt.

In der Abbildung 4.114 (mittleres Bild) wird der Such-String „MAL-L“ vorgegeben. Auch hier werden automatisch alle Treffer von BRIME gelistet. Bei diesem Beispiel ist zusätzlich die Farbgebung für Enzyme (blau) und Metabolite (grün) ersichtlich. In dem gewählten Beispiel wurde durch Auswahl von „MAL-L“ dieses in die Sammelbox eingetragen.

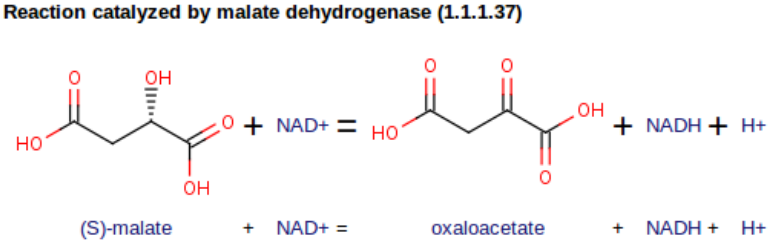

Abbildung 4.115: Die obige Reaktion wird durch die Malate Dehydrogenase katalysiert.

Zusätzlich zu der EC-Nummer 1.1.1.37 und „MAL-L“ soll eine Reaktion (siehe Abbildung 4.115) zu der Sammelbox (siehe Abbildung 4.114, rechts) hinzugefügt werden. Hierbei soll nach sämtlichen Reaktionen gesucht werden, die Malate und „NAD+“ beinhalten. Für die Reaktionssuche wird keine Autovervollständigung angeboten. Wird schließlich die Suche ausgeführt, wird der Suchbaum im Navigationsbaum für die gefundenen Treffen in der Karte aufgebaut. Der neu aufgebaute Navigationsbaum ist in Abbildung 4.116 dargestellt. Wie bereits erwähnt, sind die Suchtreffer mit der Karte verknüpft. Bei Auswahl eines Treffers werden alle gefundenen Knoten im Netzwerk hervorgehoben. Außerdem wird bei einem eindeutigen Treffer (einmalig im Netzwerk vorhanden) direkt auf den dazugehörigen Kartenabschnitt fokussiert. In dem oben genannten Beispiel wird die EC-Nummer 1.1.1.37 viermalig im Netzwerk gefunden. Die Suche nach „MAL-L“ brachte sechs Treffer und die eingegebene Teilreaktion, an der Malate und „NAD+“ beteiligt sein sollen, konnte sechs Enzymen zugeordnet werden.

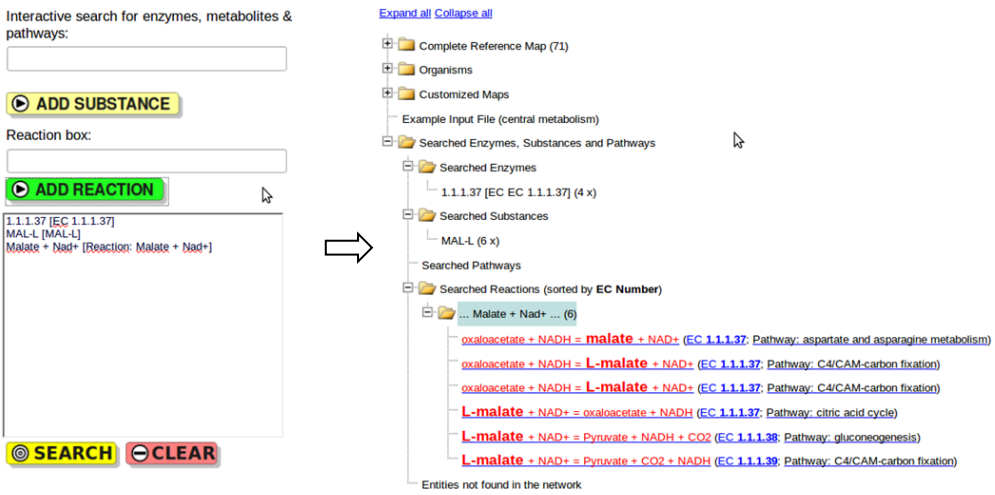

Abbildung 4.116: In der Sammelbox (linkes Bild) befinden sich Einträge für eine Suche nach dem Enzym 1.1.1.37 sowie nach dem Metaboliten MAL-L und die Teilreaktion „Malate + NAD+“. Nach Bestätigen der Suche wird der Suchbaum mit den Treffern erstellt (rechtes Bild).

4.3.7.5 Analyse von –ome Daten

BRIME bietet Anwendern eine Schnittstelle, um benutzerspezifisch Daten auf das Netzwerk abbilden zu lassen und diese in gewünschter Form zu visualisieren. Wie auch in dem Pathway-Editor MapOmnia, kann der Benutzer Datenwerte auf die Größe oder auf die Farbe der Knoten abbilden lassen. Sofern es sich um Flussdaten handelt, wird die Strichstärke der Kanten angepasst.

In diesem Anwendungsbeispiel soll gezeigt werden, wie das Abbilden von Daten aus Metabolom-Experimenten, für Enzymdaten und ebenso Flussdaten in BRIME durchgeführt wird.

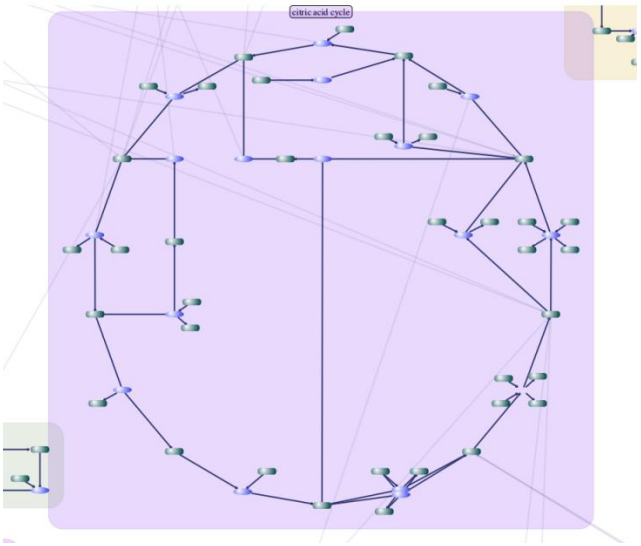

Abbildung 4.117: Es wird der Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps dargestellt.

Visualisierung und Analyse von Metabolomdaten

Zunächst sollen metabolische Daten auf das Netzwerk abgebildet werden. Abbildung 4.117 zeigt hierbei die Ausgangssituation des Netzwerkes für den Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps. Der Anwender betätigt in der Menüleiste den Menüpunkt „Upload“ und wählt das Hochladen von metabolischen Daten aus. Die in diesem Anwendungsbeispiel verwendete Datei ist im Anhang unter „Daten 4“ aufgeführt. Nach der Bestätigung des Uploads (deutsch: das Hochladen) wird der Abgleich der geladenen Daten mit dem Netzwerk durchgeführt. Es wird dem Benutzer die in Abbildung

4.118 dargestellte Ansicht gezeigt. Abbildung 4.119 zeigt den Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps. Sämtliche gefundenen Metaboliten sind hervorgehoben, nicht gefundene Knoten hingegen transparent dargestellt.

Abbildung 4.118: Auf die generischen BRENDA-Maps Karte wurden Metabolomdaten für die Visualisierung abgeglichen. Die gefundenen Metaboliten werden hervorgehoben dargestellt.

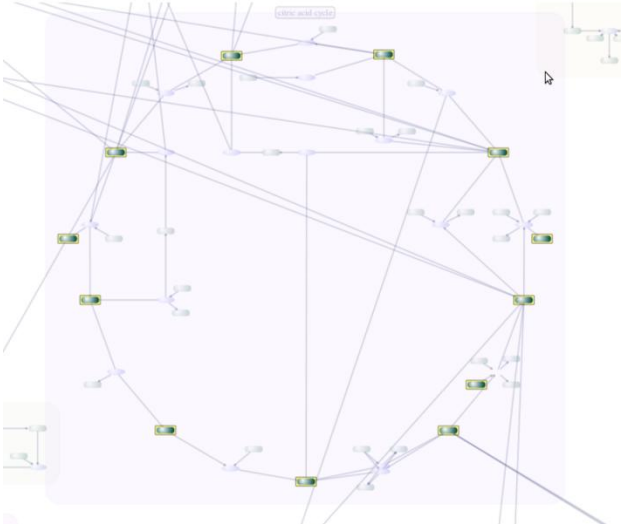

Abbildung 4.119: Es wird der Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps für einen durchgeführten Abgleich mit Metabolomdaten dargestellt.

Die zu den Metaboliten zugehörigen Datenwerte aus dem durchgeführten Daten-Upload werden im Netzwerk für den Benutzer gespeichert, um eine grafische Anpassung der Metabolit-Knoten vornehmen zu können (siehe Kapitel 4.3.6). In Abbildung 4.120 wurden die Daten mittels einer Heatmap auf die Füllfarbe der Knoten transformiert. Blau steht für niedrige Datenwerte, rot hingegen für hohe Datenwerte.

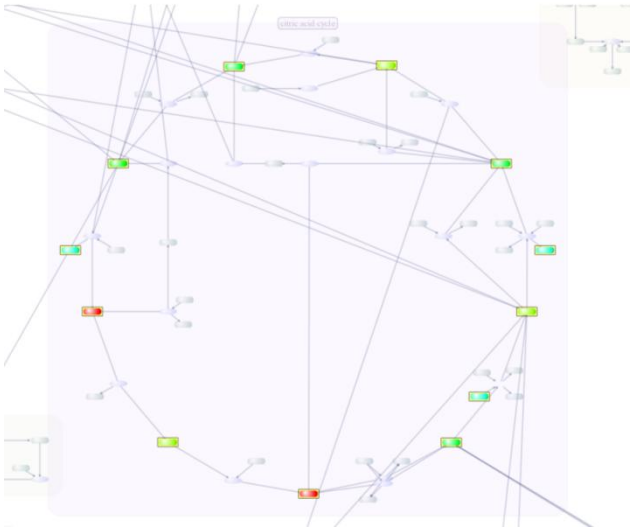

Abbildung 4.120: Auf die generische BRENDA-Maps Karte wurden Daten für die Visualisierung abgeglichen und die dazugehörigen Datenwerte mit Hilfe einer Heatmap transformiert. Es wird der Ausschnitt des Citratzyklus dargestellt.

Visualisierung und Analyse von Enzymdaten

Analog dem Vorgehen für das Hochladen für Daten aus dem vorhergegangenen Beispiel sollen in diesem Anwendungsfall Enzymdaten auf das Netzwerk abgebildet werden. Der Upload erfolgt ebenfalls über den Menüpunkt „Upload“. Die verwendete Datei ist im Anhang unter „Daten 3“ aufgeführt. Nach dem Abgleich werden die gefundenen Enzymknoten hervorgehoben dargestellt (siehe Abbildung 4.121). Abbildung 4.122 zeigt den Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps.

Abbildung 4.121: Auf die generische BRENDA-Maps Karte wurden Enzymdaten für die Visualisierung abgeglichen. Die gefundenen Enzyme werden hervorgehoben dargestellt.

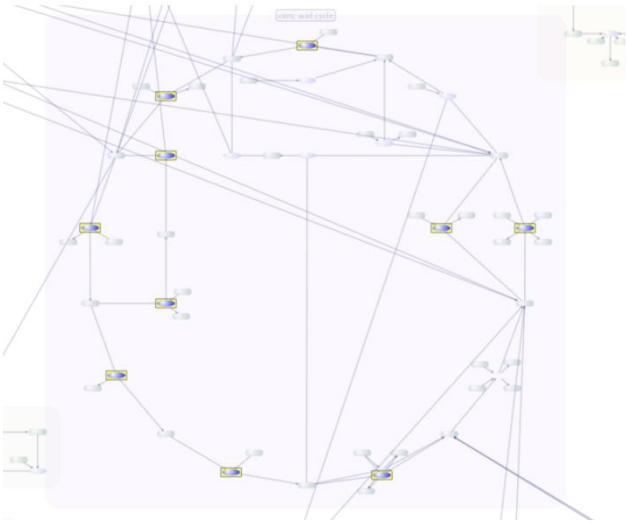

Abbildung 4.122: Es wird der Ausschnitt des Citratzyklus aus der generischen BRENDA-Maps für einen durchgeführten Abgleich mit Enzymdaten dargestellt.

Die zu den Enzymen zugehörigen experimentellen Datenwerte werden auch hier im Netzwerk benutzerspezifisch gespeichert, um eine grafische Anpassung der Enzym-Knoten vornehmen zu können. Die Daten sollen in diesem Fall größencodiert dargestellt werden.

In Abbildung 4.123 ist das Transformationsergebnis gezeigt. Kleine Knotengröße steht für niedrige Datenwerte, große hingegen für hohe Datenwerte.

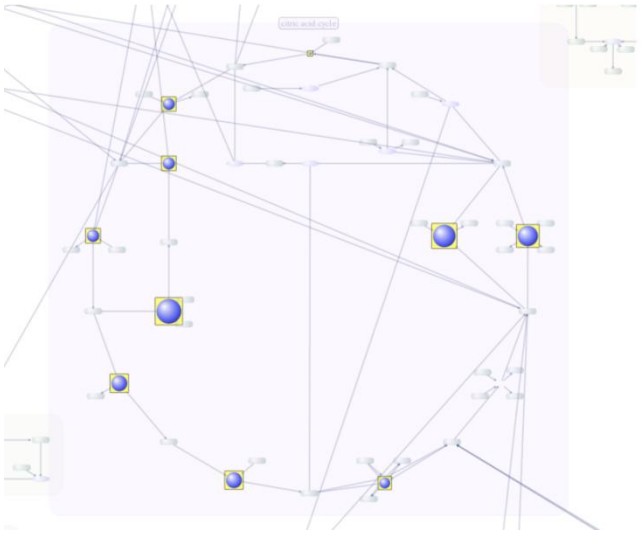

Abbildung 4.123: Auf die generischen BRENDA-Maps Karte wurden Daten für die Visualisierung abgeglichen. Hierbei wurde eine Größencodierung für die zugehörigen Datenwerte durchgeführt. Es wird der Ausschnitt des Citratzyklus dargestellt.

Visualisierung und Analyse von Flussdaten

Für die Visualisierung von Flussdaten bietet BRIME die Möglichkeit, diese Daten in Form eines CSV-Dateiformats hochzuladen. Die in diesem Beispiel verwendete Datei ist im Anhang unter „Daten 5“ aufgeführt. Wie auch bei der Visualisierung von metabolischen und Enzymdaten werden zunächst die Netzwerkobjekte identifiziert, die dem Netzwerk in BRIME zugeordnet werden können. Da die Suche reaktionsbasiert vollzogen wird, werden neben den Knoten auch die dazugehörigen Kanten gesucht. Abbildung 4.124 zeigt die Gesamtansicht der Karte, bei der die gefundenen Netzwerkobjekte hervorgehoben dargestellt werden. Abbildung 4.125 zeigt einen Ausschnitt der Karte, fokussiert auf den Stoffwechselweg Vitamin B12 Metabolismus aus dem generischen BRENDA-Maps Netzwerk. Abbildung 4.126 zeigt denselben Stoffwechselweg mit Hervorhebung der gefundenen Reaktionen.

Abbildung 4.124: Auf die generischen BRENDA-Maps Stoffwechselkarte wurden Flussdaten für die Visualisierung abgeglichen. Die gefundenen Enzyme, Metabolite und ihre verbindenden Kanten werden hervorgehoben dargestellt.

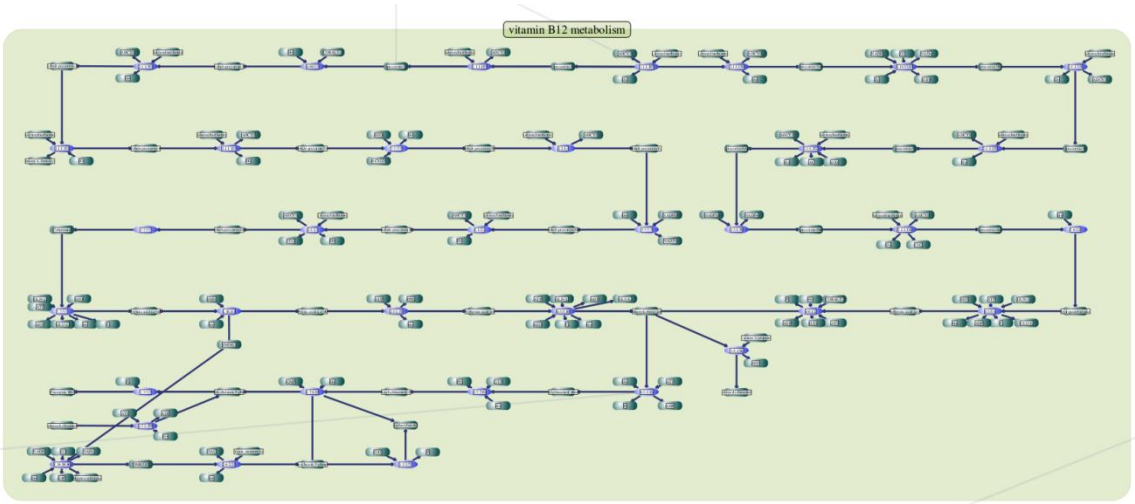

Abbildung 4.125: Es wird der Ausschnitt des Vitamin B12 Metabolismus aus der generischen BRENDA-Maps dargestellt.

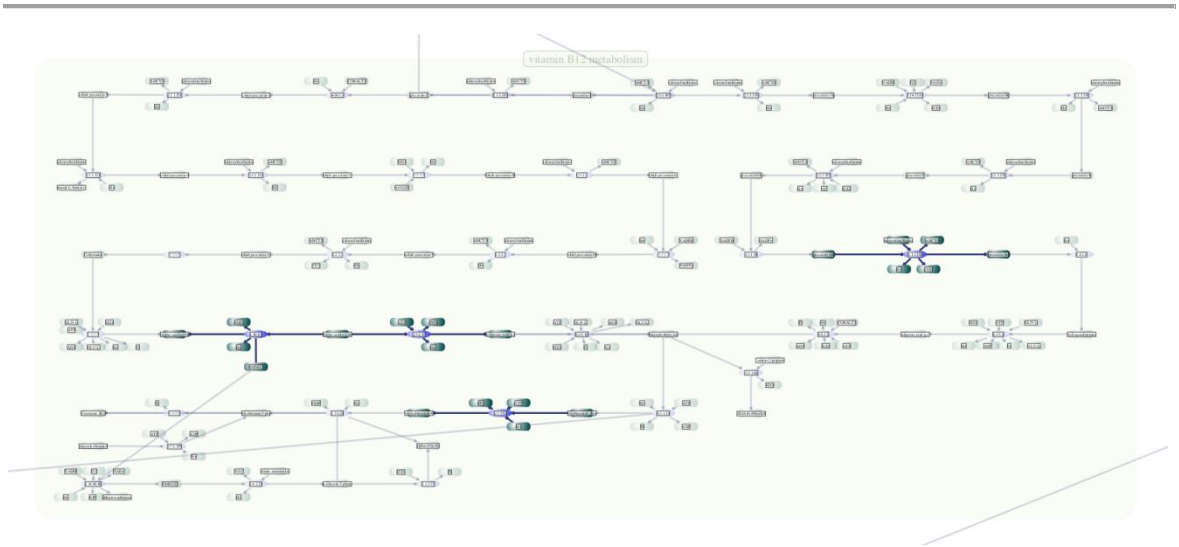

Abbildung 4.126: Es wird der Ausschnitt des Vitamin B12 Metabolismus aus der generischen BRENDA-Maps für einen durchgeführten Abgleich mit Flussdaten dargestellt.

Nachdem Flussdaten auf ein Netzwerk abgeglichen wurden, kann die Erscheinungsform der Netzwerkobjekte den Werten angepasst werden. Dieses wird in BRIME dialogbasiert durchgeführt (siehe Kapitel 4.3.5.7). In dem hier beschriebenen Beispiel wurden die Einstellungen aus dem in Abbildung 4.127 dargestelltem Dialogfenster übernommen. Das Transformationsresultat ist in Abbildung 4.128 für die Gesamtansicht gezeigt.

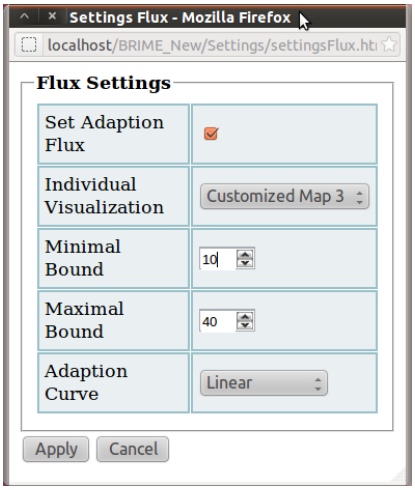

Abbildung 4.127: Über ein modales Dialogfenster bietet BRIME die Möglichkeit, Transformationen für die Codierung der Datenwerte für Flussdaten durchzuführen. Im obigen Beispiel wird für die minimale Strichstärke der Kanten 10 für die maximale 40 vorgegeben. Die Transformation soll hierbei bei der individuellen Benutzerkarte 3 durchgeführt werden.

Abbildung 4.128: Das mit Flussdaten abgeglichene Gesamtnetzwerk BRENDA-Maps. Es wurde nachträglich eine Transformation für die Codierung der Datenwerte für Flussdaten durchgeführt.

Kombination der Visualisierungen

Dem Benutzer von BRIME stehen drei Karten zur Verfügung, auf die er inviduelle Daten abbilden kann. In den vorherigen Abschnitten wurde die Visualisierung von metabolischen Daten, von Enzymdaten sowie von Flussdaten jeweils auf eine dieser individuellen Karten vollzogen. Die Abbildung verschiedener Datensätze kann auch innerhalb einer Karte durchgeführt werden. Ausgehend von dem Gesamtnetzwerk BRENDA-Maps (Teilansicht siehe Abbildung 4.129) wurde die Abbildung aller drei Datensätze auf diese Karte durchgeführt. Es ergibt sich die grafische Repräsentation aus Abbildung 4.130. Nach Bestätigung der Transformation der Knoten durch eine Größencodierung sowie der Transformation der Kanten für die Flussdarstellung zeigt sich die Grafik aus Abbildung 4.131.

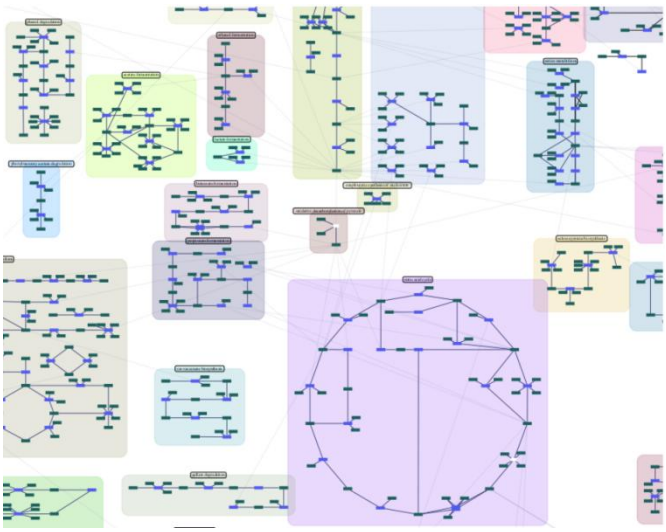

Abbildung 4.129: Auf die generische BRENDA-Maps Stoffwechselkarte sollen Datensätze von metabolischen Daten, von Enzymdaten sowie von Flussdaten für die Visualisierung abgeglichen werden. Es wird hierbei eine Teilansicht der Karte dargestellt.

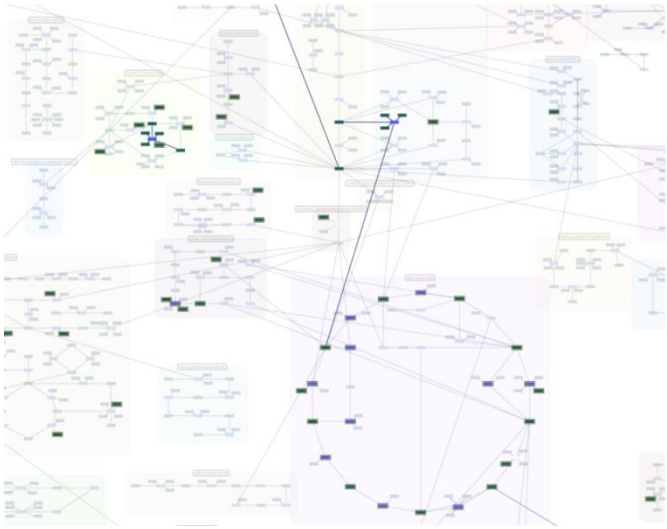

Abbildung 4.130: Ein ganzheitlicher Abgleich der Datensätze für metabolische Daten, für Enzymdaten sowie für Flussdaten wurde auf die generischen BRENDA-Maps Karte vollzogen. Es wird hierbei eine Teilansicht der Karte dargestellt.

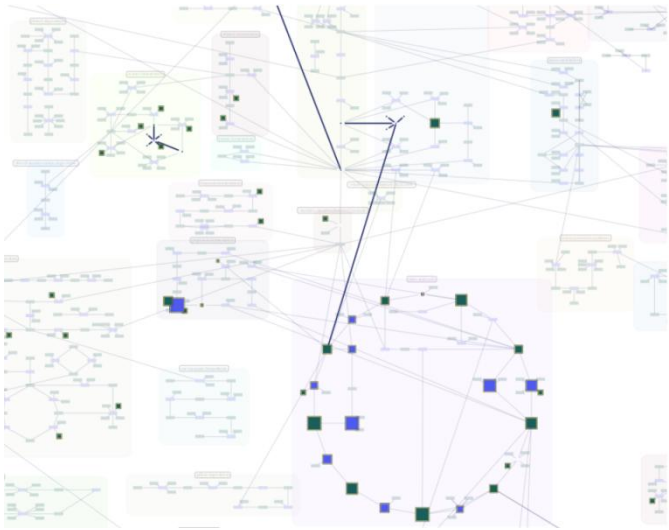

Abbildung 4.131: Auf die generische BRENDA-Maps Karte wurden Datensätze von metabolischen Daten, von Enzymdaten sowie von Flussdaten für die Visualisierung abgeglichen und die Datenwerte für eine Codierung transformiert. Es wird hierbei eine Teilansicht der Karte dargestellt.

Speicherung der Analysen

Benutzt der Anwender die Upload-Funktion für das Abbilden von Datenwerten, so werden die Analysen automatisch in Textform gespeichert und der Benutzer hat die Möglichkeit, die Analyseergebnisse auf einer Webseite anzeigen zu lassen. Die Ergebnisse können von dort als CSV-Datei gespeichert werden. Die Ergebnisdateien beinhalten zum einen den durchgeführten Abgleich mit erfolgreich gefundenen Substanzen, Enzymen oder Reaktionen und zum anderen die nicht gefundenen Einträge. Zu diesem Zweck kann jeweils eine Datei gespeichert werden. Abbildung 4.132 zeigt die Darstellung der Seite für gefundene und Abbildung 4.133 für nicht gefundene Einträge.

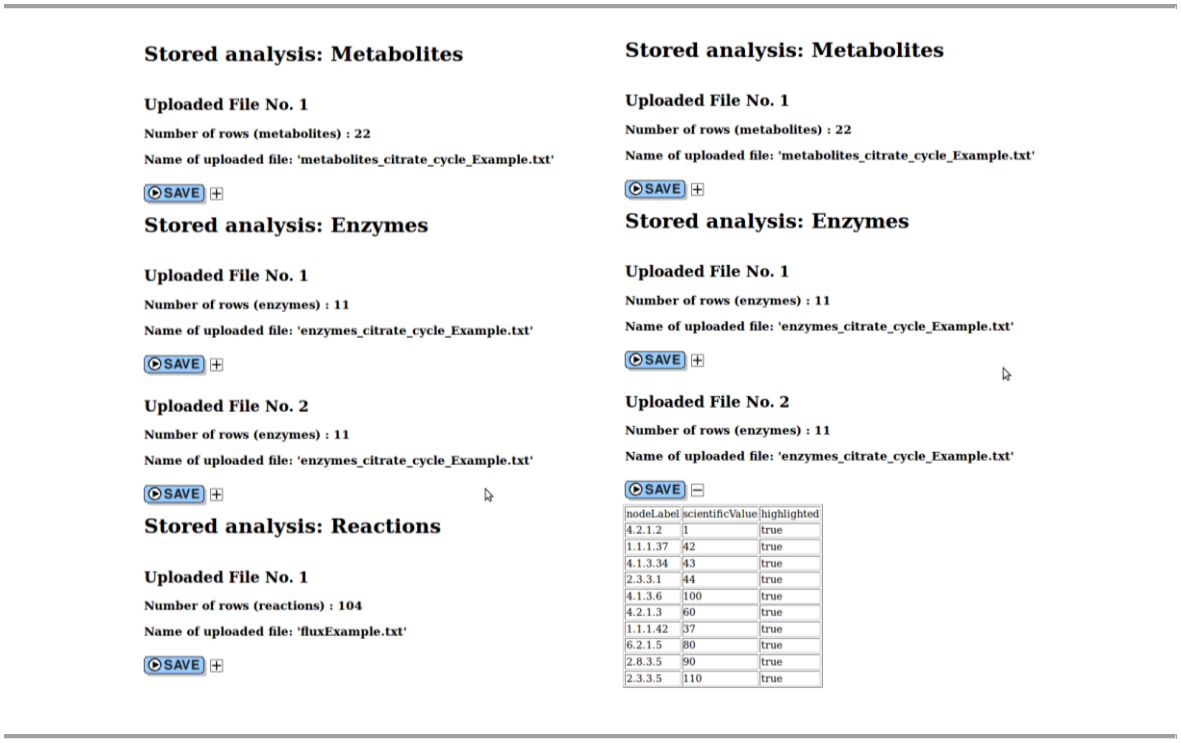

Abbildung 4.132: Die gefundenen Einträge in BRIME von hochgeladenen Dateien für Substanzen, Enzyme oder Reaktionen werden dem Anwender dargestellt und zum Download angeboten.

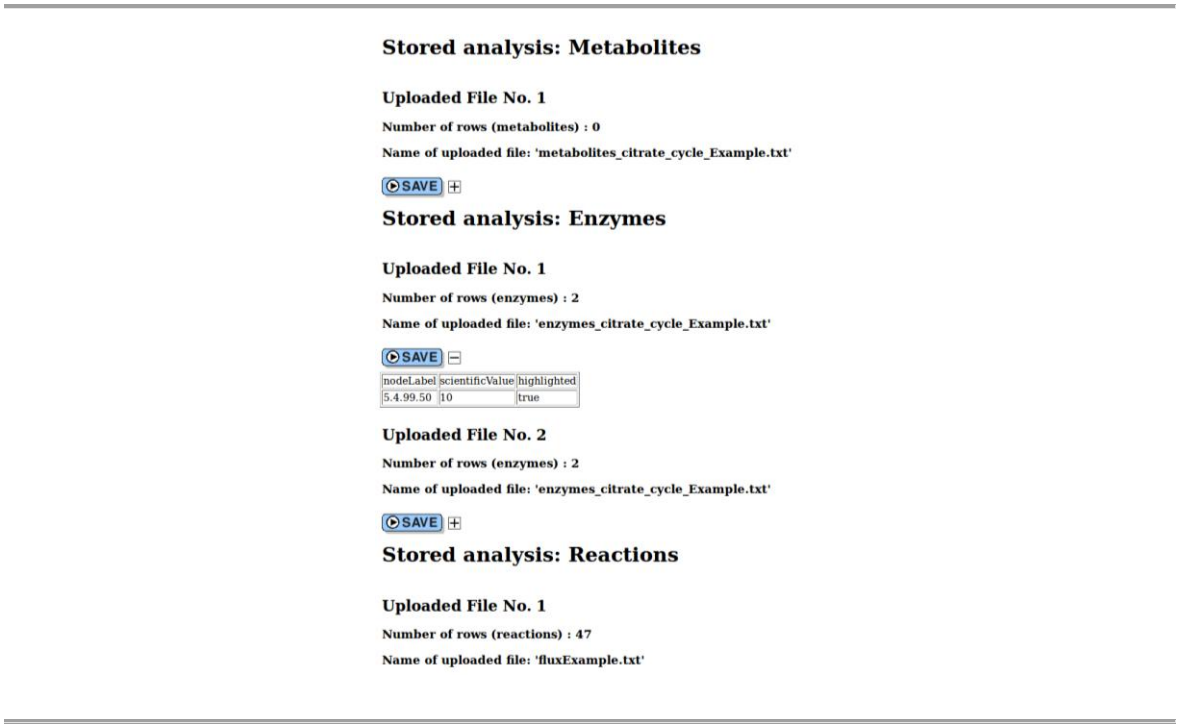

Abbildung 4.133: BRIME bietet neben den Download von gefundenen Einträgen ebenfalls den Download für nicht gefundene Einträge in BRIME.

4.3.8 Vergleich mit anderen Systemen

Bei der Gegenüberstellung von BRIME zu webbasierten Applikationen für die Darstellung metabolischer Netzwerke wird nicht angestrebt, eine vollständige Auflistung aller vorhandenen Online-Visualisierungswerkzeuge zu bieten. Vielmehr haben sich in den letzten Jahren die von KEGG angebotenen Stoffwechselkarten als Visualisierungsgrundlage etabliert. Als interaktive Webpräsenz wird zu diesem Zweck in Kapitel 4.3.8.1 kurz der KEGG Atlas vorgestellt. In Kapitel 4.3.8.2 wird der Pathway Projector als Erweiterung zu dem KEGG Atlas beschrieben. Kapitel 4.3.8.3 bietet abschließend den Vergleich der Funktionalitäten beider Webapplikationen zu BRIME.

4.3.8.1 KEGG Atlas

Der KEGG Atlas (Okuda *et al.*, 2008; Internetdokument KEGG Atlas) bietet als Portal Zugang zu den von KEGG bereitgestellten metabolischen Karten sowie zu einer Karte, die das menschliche Gehirn abbildet. Durch Nutzung des Google Web Toolkits (GWT) kann der Benutzer in den Karten hinein- und herauszoomen sowie navigieren. Auf der linken

Seite ist ein Navigationsbaum lokalisiert, der es dem Benutzer ermöglicht, Stoffwechselwege hervorzuheben. In den Tabellen in Kapitel 4.3.8.3 finden sich weiterführende Details der Funktionalitäten des KEGG Atlas. Abbildung 4.134 zeigt die Benutzeroberfläche des KEGG Atlas.

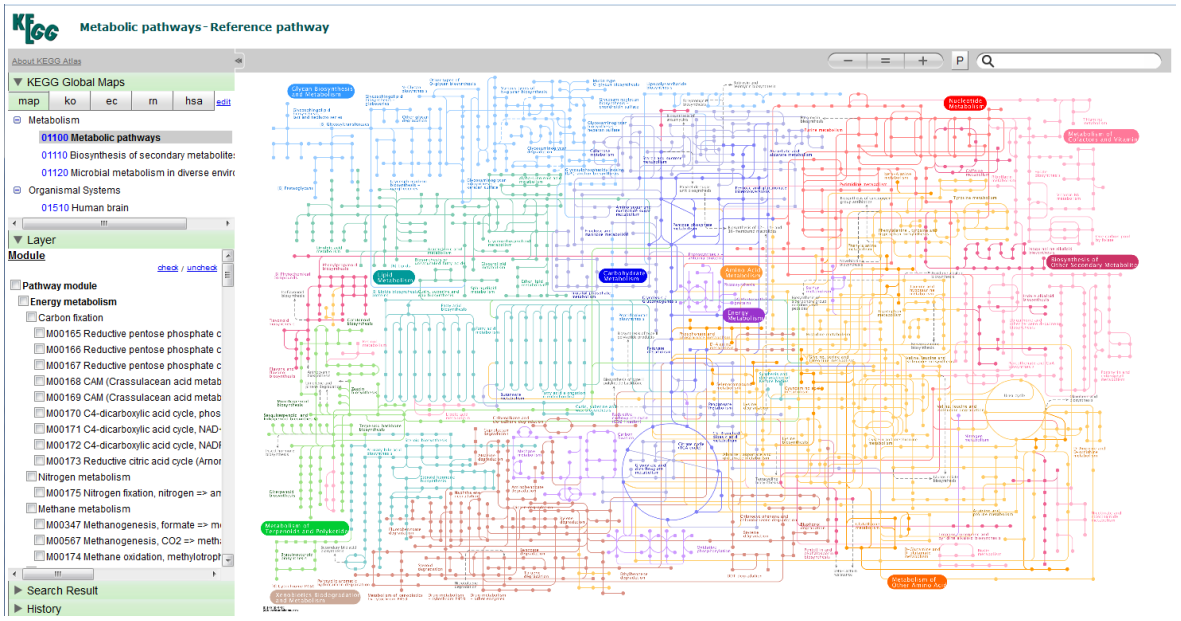

Abbildung 4.134: Benutzeroberfläche des KEGG Atlas: Die Bedienoberfläche zeigt im zentralen Bereich das ausgewählte Netzwerk. Eine Suche nach Netzwerkelementen kann in der oberen Leiste erfolgen. Auf der linken Seite kann der Nutzer unter anderem zwischen verschiedenen KEGG-Karten sowie Modulen wählen.

4.3.8.2 Pathway Projector

Der Pathway Projector (Kono *et al.*, 2009) ist als Erweiterung des KEGG Atlas zu betrachten. Es handelt sich um einen webbasierten Stoffwechselweg-Browser, der den KEGG Atlas sowie die Google Maps API benutzt, um hierbei zentrale Erweiterungen zum KEGG Atlas anzubieten. Im Gegensatz zu den KEGG-Karten werden hier ebenfalls Knoten für Enzyme präsentiert. Der Fokus der Webapplikation wird auf umfangreiche Suchfunktionalität sowie Datenzugriff gelegt. Es wird ebenfalls das Abbilden von Daten auf Netzwerkobjekte unterstützt. Weitere Funktionalitäten des Pathway Projectors werden in den Tabellen in Kapitel 4.3.8.3 gelistet. Abbildung 4.135 zeigt die Benutzeroberfläche des Pathway Projectors.

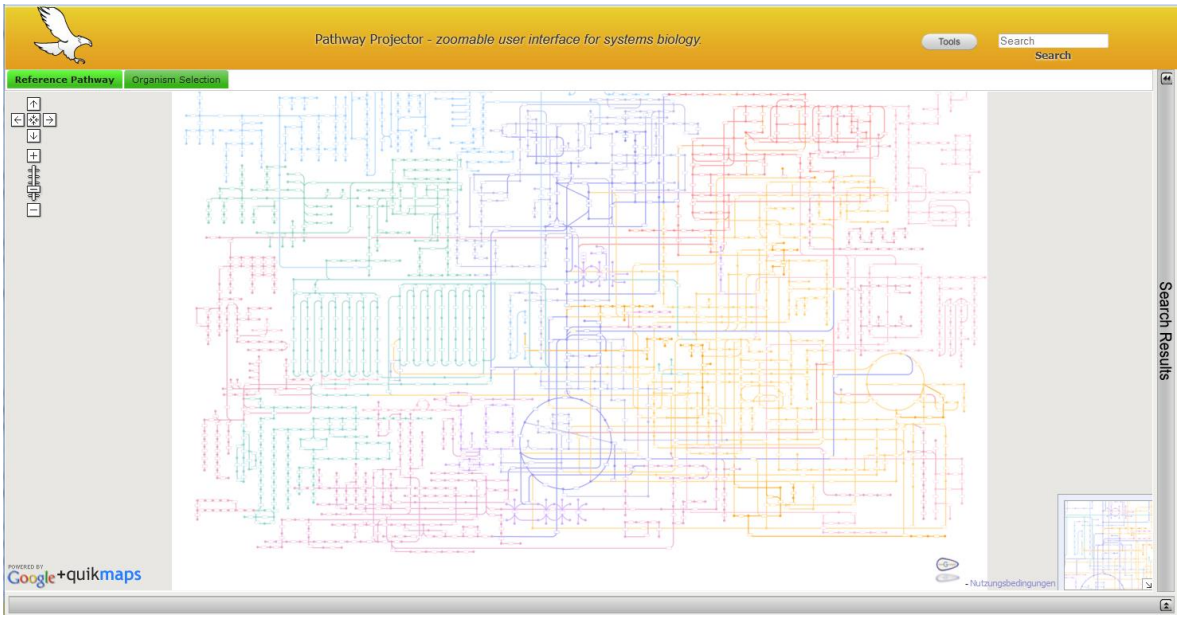

Abbildung 4.135: Benutzeroberfläche des Pathway Projectors: Im zentralen Bereich wird die Stoffwechselkarte dargestellt. Auf der linken Seite ist ein Navigationsbereich lokalisiert, im rechten unteren Bereich befindet sich eine Übersichtskarte. Im oberen Bereich der Oberfläche kann der Nutzer nach Netzwerkkomponenten wie Enzymen und Metaboliten suchen.

4.3.8.3 Vergleich zu BRIME

Zunächst werden funktionale Komponenten der webbasierten Applikationen gegenübergestellt (Unterabschnitt „Funktionale Komponenten“). Funktionale Komponenten schließen hierbei den Aufbau der Webseite, allgemein angebotene Funktionen und Export- sowie Import-Möglichkeiten ein. Anschließend wird die Performanz der Netzwerk-Browser verglichen (Unterabschnitt „Performance“). Diese wird anhand von Ladezeiten der Seite, Navigationsgeschwindigkeiten und Datenabbildung bestimmt. Die Navigation in den metabolischen Netzwerken wird im Unterabschnitt „Navigation“ näher betrachtet und im Anschluss werden die Darstellungsmöglichkeiten für diese Netzwerke erörtert (Unterabschnitt „Darstellung“). Methoden für einen Datenabgleich werden gegenübergestellt (Unterabschnitt „Abbildung von Daten“). Die Webpräsenzen werden im Unterabschnitt „Suche und Filterung“ auf vorhandene Such- und Filteroptionen von Netzwerkdaten hin untersucht. Die angebotenen Informationen zu Enzymen, Metaboliten und Stoffwechselwegen, wie eventuelle Verlinkung zu anderen

Datenbanken, oder Methoden zu Informationsgewinnung, werden im Unterabschnitt „Informationssystem“ behandelt. Abschließend folgt der Vergleich von Exportfunktionen der Analyseergebnisse sowie des Bildexports (Unterabschnitt „Speicherung und Export“).

Funktionale Komponenten

Alle hier gegenübergestellten Applikationen bieten Such- und Filteroptionen für Netzwerkelemente der metabolischen Netzwerke. Ebenso werden von allen Browsern Tooltip-Informationsfenster zu den Enzymen und Substanzen angeboten. Einen Navigationsbaum hingegen bietet nur der KEGG Atlas und BRIME. Ein zusätzliches Kontrollpanel für die Navigation innerhalb der Karte und eine kleine Übersichtskarte werden von dem Pathway Projector und BRIME, von dem KEGG Atlas hingegen nicht offeriert. Keine der Webseiten verfügt über einen Bildexport, der Export von Analyseergebnissen wird allerdings von BRIME unterstützt. Das Laden von Beispieldaten und das Abbilden von Experimentdaten können in dem Pathway Projector und BRIME erfolgen. Tabelle 4.26 gibt eine Übersicht über die Basis-Funktionalitäten des KEGG Atlas, des Pathway Projectors und von BRIME.

Tabelle 4.26: Übersicht über die Basis-Funktionalitäten des KEGG Atlas, des Pathway Projectors und von BRIME

Funktionale Komponenten	KEGG Atlas	Pathway Projector	BRIME
Navigationsbaum für Netzwerke und Netwerkobjekte	✓	⊗	✓
Kontrollpanel zur Navigationsunterstützung	⊗	✓	✓
Übersichtskarte	⊗	✓	✓
Tootip/PopUp Informationsfenster	✓	✓	✓
Suchmöglichkeiten von Netwerkelementen	✓	✓	✓
Filtermöglichkeiten von Netwerkelementen	✓	✓	✓
Export des Netzwerks als Bilddatei	⊗	⊗	⊗
Export von Analyseergebnissen	⊗	⊗	✓
Laden von Beispieldaten	⊗	✓	✓
Abbilden von Datenwerten auf Enzyme und Metaboliten	⊗	✓	✓

Performance

Die drei Webapplikationen wurden auf Kriterien wie Navigation- und Zoomverhalten, der Ladezeit von Daten für die anschließende Abbildung auf das Netzwerk und der Ladezeit beim ersten Aufruf der Webseite untersucht. Das Navigationsverhalten wurde in drei Vergrößerungsstufen des Netzwerks unterschieden, da der angezeigte Detailgrad unterschiedlich ist.

Der Vergleich der Performance der Systeme zeigt signifikante Unterschiede (Tabelle 4.27). Der Pathway Projector zeigt hierbei die größten Verzögerungszeiten. Er besitzt lange Ladezeiten bei initialem Aufruf und eine flüssige Navigation ist zu Beginn nicht möglich. Auch der KEGG Atlas weist anfangs ebenfalls eine lange Wartezeit auf. Eine anschließende Navigation in der Karte ist problemlos. Die Vergrößerung von Kartenausschnitten erfolgt schnell im KEGG Atlas. Eine Rotation der Karte wird von keinem der betrachteten Systeme angeboten. Ein wesentlicher Unterschied liegt in der Speicherung (dem Caching) von Bildausschnitten. Während BRIME jedes Bild neu lädt,

nutzen die anderen beiden Browser ein Kacheleffekt, wobei geladene Kacheln zwischengespeichert werden. Dieses bietet performante Vorteile bei fortschreitender Benutzung der Webapplikation. Das Laden einer Datei für einen Datenabgleich dauert bei dem Pathway Projector etwa 2,8-mal länger als bei BRIME mit einer Datei vergleichbarer Größe.

Tabelle 4.27: Übersicht über Kriterien für die Bestimmung performanter Unterschiede zwischen dem KEGG Atlas, dem Pathway Projector und BRIME. Die Laufzeitmessungen wurden jeweils dreimalig durchgeführt. Die Standardabweichung ist in Klammern angegeben. Mit ⊗ markierte Tabelleneinträge geben an, dass diese Funktion nicht von der Webapplikation unterstützt wird

	KEGG Atlas	**Pathway Projector**	**BRIME**
Ladezeit (erster Aufruf der Webseite)	**8 (±0,6) sec**	**20 (±1,2) sec**	**10 (±1,0) sec**
Zoom (Vergrößß auf die nächste Zoomstufe)	**<1 sec**	**5 (±0,5) sec**	**<0,5 sec**
Rotation	⊗	⊗	⊗
Navigation Nahbereich (höchste Zoomstufe)	**1,5 (±0,1) sec**	**3 (±0,3) sec**	**<0,5 sec**
Navigation Mittlerer Bereich (mittlere Zoomstufe)	**<0,5 sec**	**3 (±0,2) sec**	**<0,5 sec**
Navigation Entfernter Bereich (niedrigste Zoomstufe)	**<0,5 sec**	**<0,5 sec**	**<0,5 sec**
Datenabbildung	⊗	**20 (±2,0) sec**	**7 (±1,8) sec**

Navigation

BRIME bietet umfangreichere Navigationsmöglichkeiten im Vergleich zu den beiden Browsern. Tabelle 4.28 bietet eine Übersicht über diese Navigationsoptionen. Zusätzlich zu dem „Drag and Drop“ der Karte, welches von sämtlichen Applikationen angeboten wird, unterstützt BRIME Key-basierte Navigation mit Pfeiltasten sowie die Navigation durch einen Navigationsbaum inklusive der Fokussierung auf Stoffwechselwege. Auch kann BRIME direkt die Nachbarschaft eines Knotens fokussiert darstellen. BRIME sowie der Pathway Projector ermöglichen eine Navigation durch eine Übersichtskarte, der KEGG Atlas hingegen nicht.

Tabelle 4.28: Übersicht über Navigationsmöglichkeiten von KEGG Atlas, Pathway Projector und BRIME

Navigation	KEGG Atlas	Pathway Projector	BRIME
Key-basierte Navigation	⊗	⊗	✓
Navigation durch Übersichtskarte	⊗	✓	✓
Fokussieren auf Stoffwechselweg	⊗	⊗	✓
Drag and Drop der Karte	✓	✓	✓
Navigation durch Navigationsbaum	⊗	⊗	✓
Direkteingabe Zoom	⊗	⊗	✓
Auf Reaktion/Nachbarschaft fokussieren	⊗	⊗	✓

Darstellung

Die Darstellung des Netzwerks im Pathway Projector und in BRIME unterscheiden sich grundlegend zu der Darstellung im KEGG Atlas. Während der KEGG Atlas Hyperkanten für Enzyme benutzt, wird in den beiden anderen Browsern ein bipartiter Graph dargestellt. Alle drei Applikationen zeigen in der Netzwerkdarstellung die Lokalisation von Stoffwechselwegen. Sie bieten sowohl die Möglichkeit, eine generische Gesamtstoffwechselkarte als auch organismusspezifische Teilgraphen zu visualisieren. Individuelle Benutzerkarten bietet hingegen nur der Pathway Projector und BRIME. Eine Anzeige der Reaktionsrichtung fehlt bei den Pathway-Betrachtern KEGG Atlas und Pathway Projector. Letzteres wird nur durch BRIME unterstützt. Tabelle 4.29 gibt eine Übersicht über die Darstellungsoptionen von Netzwerken in den drei Applikationen.

Tabelle 4.29: Übersicht über die Darstellung der Netzwerke im KEGG Atlas, Pathway Projector und BRIME

Darstellung	KEGG Atlas	Pathway Projector	BRIME
Bibartiter Graph	⊗	✓	✓
Hyperkanten	✓	⊗	⊗
Gesamtstoffwechselkarte	✓	✓	✓
Organismusspezifische Teilgraphen	✓	✓	✓
Individuelle Benutzerkarten	⊗	**1**	**3**
Anzeige Reaktionsrichtung	⊗	⊗	✓
Stoffwechselwege	✓	✓	✓

Abbildung von Daten

Der KEGG Atlas bietet keine Möglichkeit, Datenwerte auf die angebotenen Stoffwechselkarten abzubilden. Der Pathway Projector und BRIME bieten diese Option. Hierbei wird von beiden Systemen das Abbilden von Daten aller „-omik“ Teildisziplinen ermöglicht. Auch das Abbilden von Flussdaten wird von beiden Applikationen unterstützt. Die nachgelagerte Transformation der Daten unter Berücksichtigung unterschiedlicher Skalen (siehe Kapitel 2.1.5.1) ist ein Alleinstellungsmerkmal von BRIME. Tabelle 4.30 listet die Optionen für das Abbilden von Daten.

Tabelle 4.30: Übersicht über das Potenzial der Webapplikationen KEGG Atlas, Pathway Projektor und BRIME für das Abbilden von Daten.

Abbildung von Daten	**KEGG Atlas**	**Pathway Projector**	**BRIME**
Abbilden von Genom Daten	⊗	✓	✓
Abbilden von Transkriptom-Daten	⊗	✓	✓
Abbilden von Proteom-Daten	⊗	✓	✓
Abbilden von Metabolom-Daten	⊗	✓	✓
Abbilden von Fluss-Daten	⊗	✓	✓
Farbcodierung	⊗	✓	✓
Größencodierung	⊗	✓	✓
Transformation der Daten	⊗	⊗	✓
Nominalskala	⊗	⊗	✓
Ordinalskala	⊗	⊗	✓
Kardinalskala	⊗	⊗	✓

Suche und Filterung

BRIME bietet umfassende Such- und Filtermöglichkeiten für metabolische Karten. Es kann hierbei sowohl nach Enzymen, Substanzen als auch nach Reaktionen und Stoffwechselwegen gesucht werden. Insbesondere die Autovervollständigung der interaktiven Suche biete eine hilfreiche Unterstützung bei Suchanfragen durch den Benutzer. Diese Funktionalität weist lediglich BRIME auf. In Tabelle 4.31 ist eine Übersicht der Such- und Filtermöglichkeiten gelistet.

Tabelle 4.31: Übersicht der Such- und Filtermöglichkeiten des KEGG Atlas, des Pathway Projector und BRIME

Suche/Filterung	KEGG Atlas	Pathway Projector	BRIME
Interaktive Suche	⊗	⊗	✓
Aufbau eines Ergebnisbaums der Suche	✓	✓	✓
Suche nach Enzymen	✓	✓	✓
Suche nach Substanzen	⊗	✓	✓
Suche nach Reaktionen	⊗	✓	✓
Suche nach Stoffwechselwegen	⊗	⊗	✓

Informationssystem

Die Darstellung der metabolischen Netzwerke repräsentiert einen hohen Informationsgehalt. Tabelle 4.32 gibt eine Übersicht über zusätzlich angebotene Informationssysteme. Weiterführende Informationen wie Moleküldarstellungen werden von allen drei Systemen offeriert. Verweise auf andere Datenbanken für Detailinformationen zu Enzymen und Substanzen und einen Verweis auf eine Reaktionsansicht bietet hingegen nur der Pathway Projector und BRIME. Die Möglichkeit Stoffwechselwege statistisch auf Vorhandensein von Enzymen zu filtern, wird ausschließlich von BRIME angeboten.

Tabelle 4.32: Übersicht über zusätzlich angebotene Informationssysteme von KEGG Atlas, Pathway Projector und BRIME

Informationssystem	KEGG Atlas	Pathway Projector	BRIME
Moleküldarstellung	✓	✓	✓
Verweis auf andere Datenbanken	⊗	✓	✓
Statistische Filterung der Stoffwechselwege	⊗	⊗	✓
Verweis auf Reaktionsansicht	⊗	✓	✓

Speicherung und Export

Lediglich die Webpräsenz BRIME bietet die Speicherung der Analyseergebnisse. Diese umfasst sowohl die Möglichkeit der Speicherung gefundener Einträge sowie die Speicherung nicht gefundener Einträge. Tabelle 4.33 zeigt die angebotenen Speicher- und Export-Eigenschaften der drei Webapplikationen.

Tabelle 4.33: Übersicht über angebotene Speicher- und Export-Eigenschaften von KEGG Atlas, Pathway Projector und BRIME

Speicherung/Export	**KEGG Atlas**	**Pathway Projector**	**BRIME**
Speicherung der Analyseergebnisse	⊗	⊗	✓
Speicherung gefundener Einträge	⊗	⊗	✓
Speicherung nicht gefundener Einträge	⊗	⊗	✓

5 Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden die in der Arbeit vorgestellten Ergebnisse zu den Einzelprojekten des Pathway-Editors MapOmnia, des TCP/IP Servers MapNet und des Webprojekt BRIME zusammengefasst und im Anschluss ein Ausblick über mögliche Erweiterungen für die Applikationen gegeben.

5.1 Zusammenfassung

5.1.1 Pathway-Editor MapOmnia

Im Rahmen dieser Doktorarbeit wurde der Pathway-Editor MapOmnia entwickelt, welcher eine umfangreiche Funktionalität besitzt. Durch den Einsatz des Dokumentierwerkzeugs Doxygen wurde zu dieser Applikation eine strukturierte Dokumentation erstellt, welche es Entwicklern ermöglicht, sich in dieses komplexe Programm einzuarbeiten und es zu erweitern.

Bei der Konzeption der Benutzeroberfläche wurde darauf geachtet, dass es in Anlehnung an verbreitete Bearbeitungsprogramme wie z. B. GIMP für die Bearbeitung von Bilddateien, oder auch OpenOffice für die Bearbeitung von Textdokumenten, die wichtigsten und gängigsten Menüelemente anwendungsfreundlich präsentiert. Dies ermöglicht einen intuitiven Umgang mit der Applikation.

MapOmnia erlaubt die Visualisierung und Bearbeitung von Netzwerken und im Speziellen die von metabolischen Netzwerken. Netzwerke bestehen in MapOmnia aus Knoten, Kanten und Gruppen. Gruppen fungieren hierbei als Superknoten, um Kanten und Knoten gruppiert darstellen zu können. Diese Umsetzung bietet die Möglichkeit, um beispielsweise Stoffwechselwege innerhalb metabolischer Netzwerke visuell abzugrenzen.

Ein Netzwerk in MapOmnia kann beliebig viele Teilnetzwerke enthalten. Diese können durch den Benutzer selektiert und separat dargestellt werden. Diese Funktionalität ermöglicht es, z.B. zu einer generischen metabolischen Gesamtkarte beliebig viele verschiedene organismusspezifische Karten zu erstellen und anzuzeigen.

Selbst bei einer hohen Anzahl von Netzwerkobjekten erfolgt der Wechsel zwischen Teilnetzwerken sehr performant. Hierfür wurde ein effizientes Datenmodell etabliert,

welches dynamisch erweiterbar ist. So können den Netzwerkobjekten neue Attribute mit zahlreich unterstützten Datenformaten zugewiesen werden. Es können z.B. Experimentdaten unterschiedlicher Zeitpunkte geladen, und jeweils als neues Attribut innerhalb von Netzwerken gespeichert werden. Die innovative „Model View“-Architektur von Qt sowie die Verwendung eines Delegates erlaubt es diese Datentypen in individueller Form zu repräsentieren und zu editieren. So öffnen sich geeignete Editoren für gewählte Attributwerte, wie z.B. ein Editor für Farben bei der Auswahl eines Farbwertes.

MapOmnia zeichnet sich durch eine hohe Interaktionsfähigkeit für erstellte Netzwerke aus. Es werden zahlreiche Selektions-, Bearbeitungs-, Navigations- und Sortier- und Filtermöglichkeiten angeboten. Selektionen von Knoten und Kanten des Netzwerks können z.B. direkt in der Netzwerkansicht selbst erfolgen oder ebenfalls in Fenstern, die der weiteren Bearbeitung der Netzwerke dienen. So ist die Baumdarstellung für die hierarchische Übersicht der Netzwerke und Netzwerkobjekte direkt mit der Darstellung des Netzwerkes gekoppelt.

Die Konstruktion von Netzwerken oder die Erweiterung von geladenen Netzwerken wird seitens des Programms in intuitiver Art und Weise unterstützt, wobei grundlegende dialogunterstützte Layout-Funktionen angeboten werden, um z.B. gesamte Graphen oder auch Teilbereiche ausrichten zu können. Es werden in MapOmnia unter anderem lineare, rasterbasierte Anordnung und Kräfte-basierte Anordnung angeboten. Auch Normalisierungsoptionen in definierbaren Teilbereichen, wie etwa die Angleichung von Abständen oder Verschieben auf nächste Rasterpunkte wird seitens des Programms unterstützt.

Durch das implementierte Import- und Export-System ist MapOmnia in der Lage viele Netzwerke aus unterschiedlichen Dateiformaten lesen und speichern zu können. Durch die Plugin-Schnittstelle können für die Applikation beliebige Import- und Exportmodule programmiert werden. Dieses ermöglicht eine weitgehende Integration bestehender Daten und den Austausch mit weiteren Programmen.

Durch die Implementierung eines Sortier- und Filter-Proxys für die Datenstruktur der Netzwerkobjekte, ist ein effizientes Werkzeug für die benutzerdefinierte Filterung verschiedener Datentypen entstanden. Ebenso können netzwerkintegrierte Daten auf- und absteigend sortiert dargestellt werden. So können geladene Netzwerke einfach und intuitiv

nach Vorhandensein von Datenstrukturen für die Netzwerkobjekt Knoten, Kanten sowie Gruppen gefiltert werden.

Es wird neben der Text-basierten Suche auch die Suche mit regulären Ausdrücken unterstützt, wodurch in Netzwerken umfassend gesucht werden kann. Hier kann z.B. einfach auf Vorhandensein von gesamten Reaktionen oder Teilreaktionen in metabolischen Netzwerken gesucht werden. Dies ermöglicht eine schnelle Identifizierung schon bekannter Reaktionen.

Der Editor weist ein breites Spektrum anwendungstypischer Funktionen auf. Eine Undo/Redo-Funktionalität, basierend auf dem Command Pattern, ermöglicht eine hohe Benutzerfreundlichkeit im Umgang mit dem Programm. Auch Funktionen wie beispielsweise „Drag and Drop“ (deutsch: Ziehen und Ablegen) sowie „Copy and Paste“ (deutsch: Kopieren und Einfügen) von einzelnen Knoten sowie gesamter Unternetzwerke wird von der Applikation unterstützt. So können komplette Reaktionen aus zwei Stoffwechselwegen kopiert und eingefügt werden.

MapOmnia ist mit einer Datenbankschnittstelle versehen, die es erlaubt Netzwerke direkt aus Datenbanken zu laden. Dies ermöglicht, Online-Ressourcen zu verwenden. Diese Schnittstelle wurde für die Stoffwechselkarte BRENDA-Maps genutzt, um das Netzwerk in MapOmnia zu visualisieren. Ebenfalls wird das Hinzuladen von Organismen als Teilnetzwerke für annotierte Organismen aus BRENDA und ebenso auch aus PATRIC, einer umfangreichen Online-Datenbank für pathogene Bakterien, ermöglicht. Das Programm bietet, neben zahlreichen Anwendungsmöglichkeiten, ein Modellierungswerkzeug, welches die Erstellung neuer Stoffwechselkarten in einfacher Weise unterstützt. Die Modellierung eines neuen Organismus wird durch Verwendung der dialoggeführten Modellierung drastisch vereinfacht. Hierdurch wird der Benutzer in die Lage versetzt, durch einfache Auswahl vorhandener Netzwerkobjekte, *in silico* Organismen zu erstellen.

Die Implementierung einer Skript-Schnittstelle erlaubt, Funktionalitäten und Methoden des Programms innerhalb von Skripten zur Laufzeit zu nutzen. Dies ermöglicht, Aufgaben zu automatisieren, Abfragen durchzuführen oder Netzwerke und Teilnetzwerke zu erstellen. Die erstellten Skripte können einfach gespeichert und abgerufen werden. So kann beispielsweise, per Skript gesteuert, eine farbliche Abhebung von speziellen

Stoffwechselwegen oder Metaboliten erfolgen. Auch das Ein- und Ausblenden von Seitenmetaboliten kann durchgeführt werden.

MapOmnia unterstützt die Visualisierung von Experimentdaten aus den „omik“ Teildisziplinen. So wird das Abbilden von Genom-, Transkriptom-, Proteom- und Metabolom-Daten und ebenso das Abbilden von Fluss-Daten ermöglicht. Eine Transformation für geladene Datensätze kann wahlweise für eine Nominal-, Ordinal- oder Kardinalskala durchgeführt werden und zwischen den Visualisierungsoptionen der Farb- sowie Größencodierung gewählt werden.

Durch die Erstellung von Animationen für Netzwerkobjekte sowie der Netzwerkszene, die die Netzwerke darstellt, kann der Anwender von MapOmnia biologische Zusammenhänge herausstellen. So lassen sich unter anderem Animationen erstellen, bei der entlang von kürzesten Pfaden innerhalb des Netzwerkes dieser Weg animiert visualisiert wird. Auch das periodische Aufblinken von Netzwerkobjekten wird ermöglicht. Dies kann beispielsweise dafür genutzt werden, um Reaktionen blinkend darzustellen, in denen ATP vorliegt. Auch Veränderungen von Metabolom-, Transkriptions- und Proteomdaten können so visualisiert werden. Sofern Flussdaten von verschiedenen Experimenten oder Fluss-Bilanz-Analysen vorliegen, kann dieser Übergang ebenfalls simuliert werden.

5.1.2 MapNet-Server

Durch Implementierung MapNet-Server wurde eine Schnittstelle etabliert, welche über das TCP/IP Netzwerkprotokoll in der Lage ist, auf die vielseitigen Anwendungsmöglichkeiten von MapOmnia zuzugreifen. Diese neu entwickelte API ermöglicht dem Benutzer z. B. eigene Webanwendungen zu erstellen, um auf Graphen online zugreifen zu können, welche auf anderen Computern im Netzwerk berechnet werden. Zu diesem Zweck wurde neben der gesamten Navigation innerhalb der Karten auch eine Upload-Funktion von Daten innerhalb dieser Schnittstelle zugänglich gemacht.

5.1.3 Webprojekt BRIME

Innerhalb der Doktorarbeit von Susanne Quester (Quester und Schomburg, 2011) wurde eine umfangreiche generische, metabolische Stoffwechselkarte erstellt. Diese enthält Enzyme und ihre katalysierten Reaktionen aus den drei wichtigsten biochemischen Datenbanken BRENDA, KEGG und MetaCyc. Dieses globale Netzwerk enthält 1667

(1347 distinkte) Enzyme sowie 5,423 (1428 distinkte) Metabolite. Das Netzwerk repräsentiert somit mehr als 1600 biochemische Reaktionen in über 120 biochemischen Stoffwechselwegen. Hierbei wurden sowohl Stoffwechselwege eingepflegt, die typisch für alle taxonomischen Gruppen sind, als auch die, welche speziell in prokariotischen Organismen vorkommen. Im Gegensatz zu den meisten existierenden globalen Stoffwechselkarten, wie z. B. die globale Stoffwechselkarte von KEGG, wurde dieses Netzwerk als bipartiter Graph entwickelt. Dies erlaubt die unabhängige Modellierung von Enzymen und Substanzen, ohne auf Hyperkanten angewiesen zu sein.

Durch BRIME wurde eine Webpräsenz geschaffen, welche Zugriff auf diese Referenzkarte ermöglicht. Neben verschiedenen interaktiven Navigation und Zoomoptionen, werden dem Benutzer von BRIME die Darstellung organismusspezifischer Stoffwechselkarten und eine Stoffwechselweg-basierte Navigation in intuitiver Art angeboten. Dank der innovativen Server-Technologie, als Backend für die aufwendigen Berechnungen und durch die Kombination mit JavaScript und PHP für das Web-Frontend, konnte eine komplett neue API entwickelt werden. Diese API basiert nicht, wie beispielsweise die vorgestellten Webapplikationen KEGG Atlas und Pathway Projector, auf AJAX/Google Maps Technologie, erlaubt dennoch eine schnelle Interaktionsmöglichkeit mit großen Netzwerken.

Zusätzlich zu der Möglichkeit biochemische Stoffwechselwege zu erkunden, kann der Anwender seine eigenen Daten analysieren. Die Enzyme und Metaboliten werden auf das Netzwerk abgebildet, welches eine visuelle Validierung von metabolischen Modellen erlaubt. Auch die Anpassung der Netzwerkobjekte durch Größen- und Farbcodierungen wird seitens BRIME unterstützt und ebenso können Flussdaten visualisiert werden. BRIME stellt eine benutzerfreundliche visuelle Schnittstelle zur Verfügung, die es erlaubt, in intuitiver Art mit Teilnetzwerken von Organismen zu interagieren, Daten zu analysieren und benutzerspezifische Karten zu erstellen.

BRIME unterstützt die Suche von Substanzen, Enzymen sowie gesamter Reaktionen und Teilreaktionen. Diese Suchfunktion erlaubt dem Nutzer, individuelle Fragestellungen zu beantworten.

Durch diese Eigenschaften stellt BRIME eine nützliche Anwendung im Bereich der metabolischen Modellierung und weiteren Bereichen der Systembiologie dar.

5.2 Ausblick

Wie bei fast jedem Softwareprojekt ist es schwer zu definieren, wann ein solches als abgeschlossen betrachtet werden kann. Insbesondere bei solch umfangreichen Projekten wie MapOmnia, kann immer neue Funktionalität implementiert oder gegebene verbessert werden. Diesem Schwerpunkt widmet sich dieses Kapitel, in dem es einen Ausblick für die Teilprojekte dieser Arbeit offeriert und aufzeigt, welche neue Funktionalität implementiert werden könnte bzw. sich schon in der Entwicklung befindet.

5.2.1 Pathway-Editor MapOmnia

MapOmnia bietet zwar eine umfangreiche Funktionalität, dennoch sind weitere Implementierungen als sinnvoll zu betrachten. Die Einbindungen weiterer Netzwerkformate, wie beispielsweise GraphML oder CellML durch die angebotene Plugin-Schnittstelle für den Datenimport und -export wäre sinnvoll, um den Anwenderkreis von MapOmnia zu vergrößern.

MapOmnia sollte benutzerspezifische Daten in größerem Umfang innerhalb der Netzwerke visualisieren können. So könnten beispielsweise in Experimenten aufgenommene Konzentrationen von Metaboliten direkt als Zeitserien auf den dazugehörigen Metabolitknoten abgebildet werden. Hier wären im Weiteren Korrelationsanalysen unter Verwendung von Korrelationsberechnungen nach Spearman oder Pearson zu ermöglichen. Dieser berechnete Rangkorrelationskoeffizient, als Eigenschaft von Netzwerkobjekten zueinander, könnte z. B. als Verbindungskante, welche größencodiert dargestellt wird, innerhalb des Netzwerkes visualisiert werden.

Auch netzwerkübergreifende Vergleiche sollten implementiert werden. So könnten phylogenetische Bäume aus den Teilnetzwerken abgeleitet werden und diese Verwandtschaftsbeziehungen ebenfalls innerhalb von MapOmnia präsentiert werden.

Eine wichtige Einbindung wäre die Berechnung von Fluss-Daten direkt innerhalb des Programms. Diese könnte für die Validierung von erstellten metabolischen Netzwerken dienen oder Experimente vorhersagen.

MapOmnia biete zwar eine Schnittstelle zu SQL-Datenbanken, dennoch könnte diese deutlich verbessert werden, indem z. B. ein eigener SQL-Browser implementiert würde, in dem SQL-Abfragen erstellt oder eigene Ansichten erzeugt werden könnten. Hierdurch

könnten Berechnungen oder auch die Kombination von Netzwerken auf SQL ausgelagert werden, aber dennoch direkt in MapOmnia visualisiert werden.
Auch würde der Anwender davon profitieren, wenn die umfangreiche Graph-Funktionalität, welche innerhalb von Boost angeboten wird, ebenfalls in das Programm integriert werden würde. Hier wären Berechnungen unterschiedlicher Layout-Funktionen, Graph-Struktur-Vergleich und Cluster-Algorithmen zu nennen.
MapOmnia sollte zusätzlich verschiedene Sprachen unterstützen. Der dynamische Baum, welcher für die Navigation zwischen geöffneten Netzwerken fungiert und die hierarchische Darstellung der Netzwerkobjekte vornimmt, bedarf einer umfassenden Weiterentwicklung und Optimierung. So sollte das Löschen und Umbenennen von Objekten auch in diesem Baum ermöglicht werden.

5.2.2 MapNet-Server

Der MapNet Server ermöglicht den Zugriff auf die Funktionen von MapOmnia, indem er auf das von Qt basierte Attribut System zugreift. Dieses ist im Server selbst aus Sicherheitsgründen gekapselt, um nur die Funktionen bereitzustellen, welche für die Netzanwendung bereitgestellt werden sollen. Hierbei handelt es sich momentan noch um einen begrenzten Umfang und MapOmnia bietet eine größere Anzahl von Anwendungsmöglichkeiten. Diese Schnittstelle wäre somit zu erweitern, um weitere Funktionen aus MapOmnia zu berücksichtigen und zukünftig anzubieten.

5.2.3 Webprojekt BRIME

Mit BRIME wurde ein System etabliert, welches durch die Verwendung des vorgestellten MapNet-Servers in der Lage ist, große Netzwerke performant als Webanwendung zu präsentieren. Da es sich um eine komplette Neuentwicklung handelt, sind hier zahlreiche Erweiterungen möglich. BRIME bietet nur Teil der Funktionalität von MapOmnia. Jedoch durch die Etablierung der generischen Schnittstelle zu dem Pathway-Editor, kann prinzipiell jede von MapOmnia bereitgestellte Funktion auch in BRIME realisiert werden. Hierdurch besteht für BRIME ein hohes Ausbaupotenzial. So könnten zukünftig weitere Analyse- oder Modellierungswerkzeuge in BRIME funktional geschaltet werden, welche schon in MapOmnia Verwendung finden. Hier wäre insbesondere die Skriptfunktionalität

zu berücksichtigen, wodurch der Anwender eigene Analysen definieren und über die bereitgestellten Schnittstellen durchführen kann.

BRIME bietet den webbasierten Zugang zu einer umfangreichen generischen Gesamtkarte sowie einer begrenzten Anzahl händisch kurierter organismusspezifischer Teilnetzwerke. Es könnten ebenfalls die in MapOmnia verfügbaren Online-Ressourcen mit angeboten oder weitere Netzwerke händisch erfasst werden, um die Verfügbarkeit von metabolischen Netzwerken diverser Organismen zu erweitern.

Glossar

Antialiasing

deutsch: die Kantenglättung.

Application Programming Interface

deutsch: die Programmierschnittstelle**.** Schnittstelle zur Anwendungsprogrammierung.

Archaea

Dritte Hauptlinie der Evolution neben den Bakterien (Bacteria) und den Eukaryoten (Eukaryota)

Blur-Effekt

deutsch: der Unschärfe-Effekt.

Bottom-Up

deutsch: Von Unten nach Oben. Arbeitsrichtungen eines Analyse-Prozesses von hoher Detailstufe zum Allgemeinen.

Brush

deutsch: der Pinsel. Ein „Brush“ ist eine farbige Fläche, welche ebenfalls eine Struktur aufweisen kann.

Button

deutsch: die Schaltfläche. Ein „Button“ ist ein Bedienelement in grafischen Benutzeroberflächen für das Aktivieren/Auslösen von Aktionen.

Callback function

deutsch: die Rückruffunktion. Die „callback function“ wird einer anderen Funktion als Parameter übergeben und unter definierten Umständen aufgerufen.

Charge

deutsch: die Ladung.

Checkboxen

deutsch: das Markierungsfeld. Eine „Checkbox“ ist ein Bedienelement in grafischen Benutzeroberflächen für die Aufnahme boolscher Datentypen.

Code Completion

deutsch: die Code Vervollständigung.

Combobox

deutsch: das Auswahlfeld. Eine „Combobox“ ist ein Bedienelement in grafischen Benutzeroberflächen für die Auswahl aus einer Liste.

Command Design Pattern

deutsch: das Kommando. Entwurfsmuster- Verhaltensmuster, Kapselung von Befehlen.

Copy and Paste

deutsch: das Kopieren und Einfügen. „Copy and Paste“ beschreibt eine Interaktionsmöglichkeit mit Applikationen. Hierbei werden zunächst Daten in einen Zwischenspeicher übertragen. Diese Daten sind nun für das Einfügen abrufbar.

Crenarchaeota

Crenarchaeota gehören zu der Domäne der Archaea.

Damping

deutsch: die Dämpfung.

Dead End Nodes

Knoten mit Knotengrad gleich 1.

Debugger

deutsch: das Fehlersuchprogramm.

Delegate

deutsch: der Stellvertreter/Vermittler.

Design Pattern

deutsch: die Entwurfsmuster. Lösungsansätze für wiederkehrende Entwurfsprobleme.

Divide and Conquer

deutsch: Teile und Herrsche. Lösungsansatz in der Informatik. Aufteilen des Problems in kleinere niedrigerer Komplexität.

Docking-Fenster

deutsch: das Andockfenster.

Drag and Drop

deutsch: Ziehen und Ablegen. „Drag and Drop“ beschreibet eine Interaktionsmöglichkeit mit Applikationen und ist eine Bedienmethode in grafischen Benutzeroberflächen für das Ziehen und Ablegen von Informationsbehafteter Bedienelementen.

Easing Function

deutsch: die Interpolationsfunktion.

Event-Handling

deutsch: die Ereignis-Behandlung.

Factory-Methoden

deutsch: die Erstellungs-Methoden.

First In–First Out

deutsch: zuerst rein – zuerst raus. Daten, die zuerst auf einen Stapel abgelegt werden, werden als erstes wieder entnommen.

Flux Balance Analysis

Mathematische Methode zur Flussuntersuchung des Metabolismus.

Font

deutsch: die Schriftart.

Frame

deutsch: der Rahmen. Teilbereich einer HTML-Seite.

Frameset

Gesamtheit der Frames auf einer HTML-Seite.

Frontend

deutsch: vorderes Ende. Entspricht in der Applikationsentwicklung einer Benutzerschnittstelle.

Handler

deutsch: das Steuerprogramm.

Highlighted

deutsch: hervorgehoben.

Hub Nodes

Knoten mit Knotengrad größer 2.

Icons

deutsch: das Symbol. Repräsentiert als Piktogramm innerhalb einer grafischen Oberfläche oft eine Datei, Verzeichnis oder eine bestimmte Aktion.

ImageMap

deutsch: die verweissensitive Grafik. Eingebettete Hyperlinks innerhalb einer Grafik.

Label

deutsch: die Beschriftung.

Last In–First Out

deutsch: zuletzt herein – zuerst hinaus. Daten, die zuletzt auf einen Stapel abgelegt werden, werden als erstes wieder entnommen.

Layout

deutsch: die Anordnung.

Linear Nodes

Knoten mit Knotengrad gleich 2.

Mammalia

deutsch: Säugetiere. Mammalia sind eine Klasse der Wirbeltiere.

Modale Fenster

Der Benutzer kann ausschließlich innerhalb des modalen Fensters arbeiten.

Model–View-Controller

deutsch: Modell-Präsentation-Steuerung.

Multiple Document Interface

Form der grafischen Benutzeroberfläche, wobei in einem Programmfenster mehrere Dokumente geöffnet werden können.

Not connected Nodes

Knoten mit Knotengrad gleich 0.

Open Source

deutsch: quelloffen. Der Quelltext der Software ist zugänglich und darf kopiert, modifiziert und verändert werden.

Parser

deutsch: der Syntaxanalysierer. Programm zur Analyse der Syntax.

Pen

deutsch: der Stift

Plugin

deutsch: das Erweiterungsmodul.

Pointer

deutsch: der Zeiger. Ein „Pointer“ entspricht einer Speicheradresse.

Property

deutsch: das Attribut.

Proxy

deutsch: der Stellvertreter.

Shortest Path

deutsch: kürzeste Wege.

Slash

deutsch: der Schrägstrich.

Slider

deutsch: der Schieber. Der „Slider“ ist ein Bedienelement in grafischen Benutzeroberflächen für die Regelung von Datenwerten.

Spring

deutsch: die Feder.

Stack

deutsch: der Stapelspeicher.

Syntax Checks

deutsch: die Überprüfung der Syntax.

Syntax Highlighting

deutsch: die Hervorhebung der Syntax.

Tag

deutsch: das Bezeichnungsschild.

Toolbar

deutsch: die Symbolleiste. Eine Toolbar ist eine Symbolleiste in grafischen Benutzeroberflächen eine einfache Interaktionsmöglichkeit mit der Applikation.

Tool-Button

deutsch: die Anwendungsschaltfläche. Ein Tool-Button ist ein Bedienelement in grafischen Benutzeroberflächen für das Auslösen von Aktionen.

Tooltip

deutsch: die Kurzinfo.

Top-Down

deutsch: Von Oben nach Unten. Arbeitsrichtungen eines Analyse-Prozesses von niedriger Detailstufe zur hoher.

Upload

deutsch: das Hochladen.

Wizard

deutsch: der Assistent. Wizards sind eine besondere Art von Eingabe-Dialogen, welche eine Sequenz von Fenstern beinhalteten. Es wird eine Schritt für Schritt Konfiguration offeriert.

Literaturverzeichnis

Akira Funahashi; Mineo Morohashi; Hiroaki Kitano; Naoki Tanimura (2003): CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. In: BIOSILICO 1 (5), S. 159–162. Online verfügbar unter http://www.sciencedirect.com/science/article/pii/S1478538203023709.

Barupal, Dinesh K.; Haldiya, Pradeep K.; Wohlgemuth, Gert; Kind, Tobias; Kothari, Shanker L.; Pinkerton, Kent E.; Fiehn, Oliver (2012): MetaMapp: mapping and visualizing metabolomic data by integrating information from biochemical pathways and chemical and mass spectral similarity. In: BMC bioinformatics 13 (1), S. 99.

Becker, Oliver (2000): X-ING: XML-Einführung. Online verfügbar unter http://www2.informatik.hu-berlin.de/~xing/Einstieg/xml1.shtml, zuletzt geprüft am 20.04.2013.

Blanchette, Jasmin; Summerfield, Mark; Ettrich, Matthias (2007): C++-GUI-Programmierung mit Qt 4. Die offizielle Einführung. München: Addison-Wesley (Programmer's choice).

Börner, Jana; Buchinger, Sebastian; Schomburg, Dietmar (2007): A high-throughput method for microbial metabolome analysis using gas chromatography/mass spectrometry. In: Anal. Biochem. 367 (2), S. 143–151.

Bosrup, Erik (2009): overLIB - Homepage. Online verfügbar unter http://www.bosrup.com/web/overlib/, zuletzt geprüft am 20.04.2013.

Busch, Melanie (2011): Verknüpfung der Reaktionen einer Stoffwechselkarte auf Datenbankebene und Visualisierung von metabolischen Flüssen im Kartenexport.

Caspi, Ron; Altman, Tomer; Dale, Joseph M.; Dreher, Kate; Fulcher, Carol A.; Gilham, Fred et al. (2010): The MetaCyc database of metabolic pathways

and enzymes and the BioCyc collection of pathway/genome databases. In: Nucleic Acids Res. 38 (Database issue), S. D473-9.

Caspi, Ron; Altman, Tomer; Dreher, Kate; Fulcher, Carol A.; Subhraveti, Pallavi; Keseler, Ingrid M. et al. (2012): The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of pathway/genome databases. In: Nucleic Acids Res. 40 (Database issue), S. D742-53.

CeDiS (Center für Digitale Systeme), Freie Universität Berlin (2005): Skalenniveaus. Freie Universität Berlin, Center für Digitale Systeme (CeDiS). Online verfügbar unter http://web.neuestatistik.de/inhalte_web/content/MOD_19216/html/comp_19264.html, zuletzt aktualisiert am 07.11.2005, zuletzt geprüft am 20.04.2013.

Covert, Markus W.; Knight, Eric M.; Reed, Jennifer L.; Herrgard, Markus J.; Palsson, Bernhard O. (2004): Integrating high-throughput and computational data elucidates bacterial networks. In: Nature 429 (6987), S. 92–96.

Cytoscape Consortium (2013): Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization. Online verfügbar unter http://www.cytoscape.org/, zuletzt aktualisiert am 19.04.2013, zuletzt geprüft am 20.04.2013.

Dawes, Beman; Abrahams, David; Rivera, Rene: Boost C++ Libraries. Online verfügbar unter http://www.boost.org/, zuletzt geprüft am 20.04.2013.

Diestel, Reinhard (2006): Graphentheorie. 3. Aufl. Berlin: Springer. Online verfügbar unter http://deposit.ddb.de/cgi-bin/dokserv?id=2784404&prov=M&dok_var=1&dok_ext=htm.

Dijkstra, E. W. (1959): A Note on Two Problems in Connexion with Graphs. In: NUMERISCHE MATHEMATIK 1 (1), S. 269–271. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.7577.

Dr. Klaus Kriegel (2010): Breitensuche. Warteschlangen und die Grundidee der Breitensuche. Online verfügbar unter http://www.inf.fu-

berlin.de/lehre/SS10/infb/dfs.pdf, zuletzt aktualisiert am 29.04.2010, zuletzt geprüft am 20.04.2013.

Eckstein, Silke (2011): Informationsmanagement in der Systembiologie. Datenbanken, Integration, Modellierung. Berlin: Springer.

Elektronik-Kompendium: TCP/IP. Online verfügbar unter http://www.elektronik-kompendium.de/sites/net/0606251.htm, zuletzt geprüft am 20.04.2013.

Fiehn, Oliver (2002): Metabolomics--the link between genotypes and phenotypes. In: Plant Mol. Biol. 48 (1-2), S. 155–171.

Flanagan, David (2007): JavaScript. Das umfassende Referenzwerk. 3. Aufl. s.l: O'Reilly Verlag. Online verfügbar unter http://ebooks.ciando.com/book/index.cfm/bok_id/11277.

Foundation, Eclipse; Inc: Eclipse - The Eclipse Foundation open source community website. Online verfügbar unter http://www.eclipse.org/, zuletzt geprüft am 20.04.2013.

Free Software Foundation (FSF): GSL - GNU Scientific Library - GNU Project - Free Software Foundation (FSF). Online verfügbar unter http://www.gnu.org/software/gsl/, zuletzt geprüft am 20.04.2013.

Friendly, Michael (1995): Milestones in the history of thematic cartography, statistical graphics, and data visualization. In: 13TH INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA 2002), AIX EN PROVENCE, S. 59–66. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.190.9498.

Funahashi, A.; Matsuoka, Y.; Jouraku, A.; Morohashi, M.; Kikuchi, N.; Kitano, H. (2008): CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. In: Proc. IEEE 96 (8), S. 1254–1265.

Gillespie, J. J.; Wattam, A. R.; Cammer, S. A.; Gabbard, J. L.; Shukla, M. P.; Dalay, O. et al. (2011): PATRIC: the Comprehensive Bacterial Bioinformatics

Resource with a Focus on Human Pathogenic Species. In: Infection and Immunity 79 (11), S. 4286–4298.

Harold, Elliotte Rusty (2002): Die XML-Bibel. [XML-Einführung: alles zu Elementen, Tags, Attributen, DTD und Namensräumen ; beherrschen Sie die ganze Power von CSS und XSL ; reizen Sie die XML-Möglichkeiten voll aus mit XLinks, XPointer, Schemas, SVG und XHTML ; CD-ROM: XML-Browser und -Werkzeuge, XML-W3C-Standards, Programmiercode]. 2. Aufl. Bonn: Mitp.

Heymann, Stefan (2003): XML DTD. Online verfügbar unter http://www.stefanheymann.de/xml/dtdxml.htm, zuletzt aktualisiert am 28.10.2010, zuletzt geprüft am 20.04.2013.

Internetdokument DTD: DTD Tutorial. Online verfügbar unter http://www.w3schools.com/dtd/, zuletzt geprüft am 20.04.2013.

Internetdokument Enzyme Database - BRENDA (2013): Enzyme Database - BRENDA. Online verfügbar unter http://www.brenda-enzymes.info/, zuletzt aktualisiert am 01.01.2013, zuletzt geprüft am 20.04.2013.

Internetdokument HTML 4.01-Spezifikation: Einführung in HTML 4. Online verfügbar unter http://www.edition-w3.de/TR/1999/REC-html401-19991224/intro/intro.html, zuletzt geprüft am 20.04.2013.

Internetdokument KEGG Atlas: KEGG Atlas. Online verfügbar unter http://www.genome.jp/kegg/atlas.html, zuletzt geprüft am 20.04.2013.

Internetdokument PHP: PHP: Was ist PHP? - Manual. Online verfügbar unter http://www.php.net/manual/de/intro-whatis.php, zuletzt geprüft am 20.04.2013.

Internetdokument Programmiersprachen (2013): Programmiersprachen. Online verfügbar unter http://de.kioskea.net/contents/langages/langages.php3#top, zuletzt aktualisiert am 01.04.2013, zuletzt geprüft am 20.04.2013.

Internetdokument SBML (2013): Main Page - SBML.org. Online verfügbar unter http://sbml.org/Main_Page, zuletzt aktualisiert am 28.01.2013, zuletzt geprüft am 20.04.2013.

Internetdokument Skalenniveaus (2005): Skalenniveaus. Online verfügbar unter http://web.neuestatistik.de/inhalte_web/content/MOD_19216/html/comp_19264.html, zuletzt aktualisiert am 07.11.2005, zuletzt geprüft am 20.04.2013.

Internetdokument Wiki Doxygen: Doxygen › Wiki › ubuntuusers.de. Online verfügbar unter http://wiki.ubuntuusers.de/Doxygen, zuletzt geprüft am 20.04.2013.

Junker, Björn H.; Klukas, Christian; Schreiber, Falk (2006): VANTED: a system for advanced data analysis and visualization in the context of biological networks. In: BMC Bioinformatics 7, S. 109.

Kanehisa, Minoru (2009): Representation and analysis of molecular networks involving diseases and drugs. In: Genome Inform 23 (1), S. 212–213.

Kanehisa, Minoru; Araki, Michihiro; Goto, Susumu; Hattori, Masahiro; Hirakawa, Mika; Itoh, Masumi et al. (2008): KEGG for linking genomes to life and the environment. In: Nucleic Acids Res. 36 (Database issue), S. D480-4.

Kanehisa, Minoru; Goto, Susumu; Furumichi, Miho; Tanabe, Mao; Hirakawa, Mika (2010): KEGG for representation and analysis of molecular networks involving diseases and drugs. In: Nucleic Acids Res. 38 (Database issue), S. D355-60.

Kanehisa Laboratories 1: KEGG: Kyoto Encyclopedia of Genes and Genomes. Online verfügbar unter http://www.genome.jp/kegg/, zuletzt geprüft am 20.04.2013.

Kanehisa Laboratories 2 (2010): KGML (KEGG Markup Language). Online verfügbar unter http://www.kegg.jp/kegg/xml/, zuletzt geprüft am 20.04.2013.

Kanehisa Laboratories 3: KEGG PATHWAY Database. Online verfügbar unter http://www.genome.jp/kegg/pathway.html, zuletzt geprüft am 20.04.2013.

Kitano, Hiroaki (2002): Computational systems biology. In: Nature 420 (6912), S. 206–210.

Klukas, Christian (2012): VANTED - Visualization and Analysis of Networks containing Experimental Data. Online verfügbar unter http://vanted.ipk-gatersleben.de/, zuletzt geprüft am 20.04.2013.

Kohl, Michael; Wiese, Sebastian; Warscheid, Bettina (2011): Cytoscape: software for visualization and analysis of biological networks. In: Methods Mol. Biol. 696, S. 291–303.

Kono, Nobuaki; Arakawa, Kazuharu; Ogawa, Ryu; Kido, Nobuhiro; Oshita, Kazuki; Ikegami, Keita et al. (2009): Pathway projector: web-based zoomable pathway browser using KEGG atlas and Google Maps API. In: PLoS ONE 4 (11), S. e7710.

Kremling, Andreas (2011): Kompendium Systembiologie. Mathematische Modellierung und Modellanalyse. 1. Aufl. s.l: Vieweg+Teubner (GWV). Online verfügbar unter http://ebooks.ciando.com/book/index.cfm/bok_id/328903.

Krömer, Jens Olaf; Fritz, Michel; Heinzle, Elmar; Wittmann, Christoph (2005): In vivo quantification of intracellular amino acids and intermediates of the methionine pathway in Corynebacterium glutamicum. In: Anal. Biochem. 340 (1), S. 171–173.

Kuhlins, Stefan; Schader, Martin (2005): Die C++-Standardbibliothek. Einführung und Nachschlagewerk ; mit 37 Tabellen. 4. Aufl. Berlin: Springer. Online verfügbar unter http://deposit.ddb.de/cgi-bin/dokserv?id=2614516&prov=M&dok_var=1&dok_ext=htm.

Kunze, Ralf (2001): XML Parser. Online verfügbar unter http://www-lehre.inf.uos.de/~rkunze/flashweather/Diplomarbeit/node20.html, zuletzt aktualisiert am 04.04.2003, zuletzt geprüft am 20.04.2013.

Leiserson, Charles E.; Stein, Clifford; Cormen, Thomas H. (2007): Algorithmen - eine Einführung. 2. Aufl. München: Oldenbourg.

Lin, Jimmy; Qian, Jiang (2007): Systems biology approach to integrative comparative genomics. In: Expert Rev Proteomics 4 (1), S. 107–119.

Louis, Dirk; Müller, Peter (2007): Das Java 6 codebook. München [u.a.]: Addison-Wesley.

Matthiesen, Günter; Unterstein, Michael (2003): Relationale Datenbanken und SQL. [Konzepte der Entwicklung und Anwendung]. 3. Aufl. München ;, Boston [u.a.]: Addison-Wesley.

MDL Information Systems (2001): MDL-Mol-File. Online verfügbar unter http://www2.chemie.uni-erlangen.de/services/vsc/db_vsc/remarks/mol_subst.html, zuletzt aktualisiert am 27.09.2001, zuletzt geprüft am 20.04.2013.

Merkl, Rainer; Waack, Stephan (2009): Bioinformatik interaktiv. Grundlagen, Algorithmen, Anwendungen. 2. Aufl. Weinheim: Wiley-VCH.

Misch, Peter: Grundlagen der Informatik.

Molkentin, Daniel (2006): Qt 4. Einführung in die Applikationsentwicklung. München: Open Source Press.

Murrmann, Daniel (2008): Datenvisualisierung. Informationsvisualisierung von Datenbeständen. Saarbrücken: Vdm Verl. Dr. Müller.

Noack, Wilhelm (2010): PHP. Grundlagen - Erstellung dynamischer Web-Seiten. 9., veränd. Aufl., September 2010. Hannover.

Noack, Wilhelm (2011): JavaScript. Eine Einführung. 7., unveränd. Aufl., August 2011. Hannover (RRZN-Handbuch).

Nokia Corporation (2013): Qt - Cross-platform application and UI framework — Qt - A cross-platform application and UI framework. Online verfügbar unter http://qt.nokia.com/, zuletzt aktualisiert am 20.04.2013, zuletzt geprüft am 20.04.2013.

Nokia Corporation (2013): Qt Creator IDE and tools — Qt - A cross-platform application and UI framework. Online verfügbar unter http://qt.nokia.com/products/developer-tools/, zuletzt aktualisiert am 20.04.2013, zuletzt geprüft am 20.04.2013.

O'Farrell, P. H. (1975): High resolution two-dimensional electrophoresis of proteins. In: J. Biol. Chem. 250 (10), S. 4007–4021.

Okuda, Shujiro; Yamada, Takuji; Hamajima, Masami; Itoh, Masumi; Katayama, Toshiaki; Bork, Peer et al. (2008): KEGG Atlas mapping for global analysis of metabolic pathways. In: Nucleic Acids Res. 36 (Web Server issue), S. W423-6.

Oliver, S. G.; Winson, M. K.; Kell, D. B.; Baganz, F. (1998): Systematic functional analysis of the yeast genome. In: Trends Biotechnol. 16 (9), S. 373–378.

Orth, Jeffrey D.; Thiele, Ines; Palsson, Bernhard Ø. (2010): What is flux balance analysis? In: Nat Biotechnol 28 (3), S. 245–248.

Pavlopoulos, Georgios A.; Wegener, Anna-Lynn; Schneider, Reinhard (2008): A survey of visualization tools for biological network analysis. In: BioData Min 1, S. 12.

Punin, John: XGMML (eXtensible Graph Markup and Modeling Language). Online verfügbar unter http://www.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/, zuletzt geprüft am 20.04.2013.

Quester, Susanne; Schomburg, Dietmar (2011): EnzymeDetector: an integrated enzyme function prediction tool and database. In: BMC Bioinformatics 12, S. 376.

QuinStreet Inc: What is binary format? - A Word Definition From the Webopedia Computer Dictionary. Online verfügbar unter http://www.webopedia.com/TERM/B/binary_format.html, zuletzt geprüft am 20.04.2013.

Rathmann, Uwe; Wilgen, Josef (2011): Qwt User's Guide: Qwt - Qt Widgets for Technical Applications. Online verfügbar unter http://qwt.sourceforge.net/, zuletzt aktualisiert am 01.08.2011, zuletzt geprüft am 20.04.2013.

RRZN (2011): C++ für C-Programmierer. Begleitmaterial zu Vorlesungen/Kursen. 15., unveränd. Aufl., Januar 2011. Hannover: RRZN.

Scheer, Maurice; Grote, Andreas; Chang, Antje; Schomburg, Ida; Munaretto, Cornelia; Rother, Michael et al. (2011): BRENDA, the enzyme information system in 2011. In: Nucleic Acids Res. 39 (Database issue), S. D670-6.

Shannon, Paul; Markiel, Andrew; Ozier, Owen; Baliga, Nitin S.; Wang, Jonathan T.; Ramage, Daniel et al. (2003): Cytoscape: a software environment for integrated models of biomolecular interaction networks. In: Genome Res. 13 (11), S. 2498–2504.

Smoot, Michael E.; Ono, Keiichiro; Ruscheinski, Johannes; Wang, Peng-Liang; Ideker, Trey (2011): Cytoscape 2.8: new features for data integration and network visualization. In: Bioinformatics 27 (3), S. 431–432.

Snyder, E. E.; Kampanya, N.; Lu, J.; Nordberg, E. K.; Karur, H. R.; Shukla, M. et al. (2007): PATRIC: the VBI PathoSystems Resource Integration Center. In: Nucleic Acids Res. 35 (Database issue), S. D401-6.

SRI International (2011): MetaCyc Encyclopedia of Metabolic Pathways. Online verfügbar unter http://metacyc.org/, zuletzt geprüft am 20.04.2013.

Stocker, Christian (2008): LiveSearch - Wiki Blog - Public Liip Wiki. Online verfügbar unter https://fosswiki.liip.ch/display/BLOG/LiveSearch, zuletzt geprüft am 20.04.2013.

Stroustrup, Bjarne (2009): Die C++-Programmiersprache. 4. Aufl. München [u.a.]: Addison Wesley.

The Apache Software Foundation (2012): About the Apache HTTP Server Project - The Apache HTTP Server Project. Online verfügbar unter http://httpd.apache.org/ABOUT_APACHE.html, zuletzt aktualisiert am 11.05.2012, zuletzt geprüft am 20.04.2013.

The Apache Software Foundation (2013): Welcome! - The Apache HTTP Server Project. Online verfügbar unter http://httpd.apache.org/, zuletzt aktualisiert am 25.02.2013, zuletzt geprüft am 20.04.2013.

The GIMP Team: GIMP - The GNU Image Manipulation Program. Online verfügbar unter http://www.gimp.org/, zuletzt geprüft am http://www.celldesigner.org/.

The Systems Biology Institute: CellDesigner. Online verfügbar unter http://www.celldesigner.org/, zuletzt geprüft am 20.04.2013.

Tittmann, Peter (2003): Graphentheorie. Eine anwendungsorientierte Einführung ; mit zahlreichen Beispielen und 80 Aufgaben. München: Fachbuchverl. Leipzig im Hanser-Verl (Mathematik-Studienhilfen).

Universität Passau: GML. Online verfügbar unter http://www.fim.uni-passau.de/en/fim/faculty/chairs/theoretische-informatik/projects.html, zuletzt geprüft am 20.04.2013.

van Heesch, Dimitri (2013): Doxygen. Online verfügbar unter http://www.stack.nl/~dimitri/doxygen/, zuletzt aktualisiert am 20.01.2013, zuletzt geprüft am 20.04.2013.

Virginia Bioinformatics Institute: PATRIC. Online verfügbar unter http://www.patricbrc.org/portal/portal/patric/Home, zuletzt geprüft am 20.04.2013.

Volkmann, Lutz (2011): Graphen an allen Ecken und Kanten. zweite Version. Aachen. Online verfügbar unter http://www.math2.rwth-aachen.de/files/gt/buch/graphen an allen ecken und kanten.pdf.

W3C Communications Team: Einführung in HTML 4. Online verfügbar unter http://www.edition-w3.de/TR/1999/REC-html401-19991224/intro/intro.html, zuletzt geprüft am 20.04.2013.

W3C Communications Team (2001): XML in 10 points. Online verfügbar unter http://www.w3.org/XML/1999/XML-in-10-points.html, zuletzt aktualisiert am 25.01.2011, zuletzt geprüft am 20.04.2013.

Webb, Edwin C. (1992): Enzyme nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the nomenclature and classification of enzymes. San

Diego: Published for the International Union of Biochemistry and Molecular Biology by Academic Press.

Wendisch, Volker F.; Bott, Michael; Kalinowski, Jörn; Oldiges, Marco; Wiechert, Wolfgang (2006): Emerging Corynebacterium glutamicum systems biology. In: J. Biotechnol. 124 (1), S. 74–92.

Wieland, Thomas (2001): Grundlagen der objektorientierten Programmierung in C++. Online verfügbar unter http://www.cpp-entwicklung.de/cpplinux/cpp_main/node4.html, zuletzt aktualisiert am 31.01.2002, zuletzt geprüft am 20.04.2013.

Wiki des Lehrstuhls BISOR (2011): Dijkstra Algorithmus – Operations-Research-Wiki. Online verfügbar unter https://bisor.wiwi.uni-kl.de/orwiki/index.php/Dijkstra_Algorithmus, zuletzt aktualisiert am 27.04.2011, zuletzt geprüft am 20.04.2013.

Wikipedia (Hg.) (2013): Binärdatei. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=104988506, zuletzt aktualisiert am 02.04.2013, zuletzt geprüft am 20.04.2013.

Wikipedia (Hg.) (2013): Initialisierungsdatei. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=105262907, zuletzt aktualisiert am 03.04.2013, zuletzt geprüft am 20.04.2013.

Wikipedia (Hg.) (2013): Datenbank. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=105842438, zuletzt aktualisiert am 16.04.2013, zuletzt geprüft am 20.04.2013.

Wirtz, Markus; Nachtigall, Christof (2006): Deskriptive Statistik. 4. Aufl. Weinheim: Juventa-Verl (Statistische Methoden für Psychologen, 1).

World Wide Web Consortium (2013): World Wide Web Consortium (W3C). Online verfügbar unter http://www.w3.org/, zuletzt aktualisiert am 17.04.2013, zuletzt geprüft am 20.04.2013.

Yahoo!: YUI 2 — Yahoo! User Interface Library. Online verfügbar unter http://developer.yahoo.com/yui/2/, zuletzt geprüft am 20.04.2013.

Zippel, Uwe (2008): XML-Parser Grundlegendes. Online verfügbar unter http://www.uzi-web.de/parser/parser_grundlegendes.htm, zuletzt geprüft am 20.04.2013.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

QVariant

Tabelle A.1: Übersicht der von *QVariant* unterstützen Datentypen. (entnommen aus der Qt Hilfe der Qt Dokumentation; Nokia Corporation, 2013a)

Konstante	Datentyp
QVariant::Invalid	Kein Typ
QVariant::BitArray	QBitArray
QVariant::Bitmap	QBitmap
QVariant::Bool	bool
QVariant::Brush	QBrush
QVariant::ByteArray	QByteArray
QVariant::Char	QChar
QVariant::Color	QColor
QVariant::Cursor	QCursor
QVariant::Date	QDate
QVariant::DateTime	QDateTime
QVariant::Double	double
QVariant::EasingCurve	QEasingCurve
QVariant::Font	QFont
QVariant::Hash	QVariantHash
QVariant::Icon	QIcon
QVariant::Image	QImage
QVariant::Int	an int
QVariant::KeySequence	QKeySequence
QVariant::Line	QLine
QVariant::LineF	QLineF
QVariant::List	QVariantList
QVariant::Locale	QLocale

QVariant::LongLong	qlonglong
QVariant::Map	QVariantMap
QVariant::Matrix	QMatrix(obsolete)
QVariant::Transform	QTransform
QVariant::Matrix4x4	QMatrix4x4
QVariant::Palette	QPalette
QVariant::Pen	QPen
QVariant::Pixmap	QPixmap
QVariant::Point	QPoint
QVariant::PointArray	QPointArray
QVariant::PointF	QPointF
QVariant::Polygon	QPolygon
QVariant::Quaternion	QQuaternion
QVariant::Rect	QRect
QVariant::RectF	QRectF
QVariant::RegExp	QRegExp
QVariant::Region	QRegion
QVariant::Size	QSize
QVariant::SizeF	QSizeF
QVariant::SizePolicy	QSizePolicy
QVariant::String	QString
QVariant::StringList	QStringList
QVariant::TextFormat	QTextFormat
QVariant::TextLength	QTextLength
QVariant::Time	QTime
QVariant::UInt	uint
QVariant::ULongLong	qulonglong
QVariant::Url	QUrl
QVariant::Vector2D	QVector2D

QVariant::Vector3D	QVector3D
QVariant::Vector4D	QVector4D
QVariant::UserType	Benutzer definiert.

Verwendete Dateien MapOmnia

Daten 1: example_organism.csv.

```
nodeLabel               value
CO2                     10
ADP                     40
H2O                     100
Spermine                30
propionyl-CoA           15
glutathione             67
dolichyl phosphate      145
NADPH                   168
```

Daten 2: example_flux.csv.

```
nodeLabel   value
2.5.1.11    10
2.5.1.21    40
2.5.1.62    100
2.7.1.21    30
2.7.1.35    15
2.7.8.13    67
2.7.7.63    145
```

Verwendete Dateien BRIME

Daten 3: enzymes_citrate_cycle_Example.txt.

nodeLabel	scientificValue	highlighted
4.2.1.2	1	true
1.1.1.37	42	true
4.1.3.34	43	true
2.3.3.1	44	true
4.1.3.6	100	true
4.2.1.3	60	true
1.1.1.42	37	true
6.2.1.5	80	true
2.8.3.5	90	true
5.4.99.50	10	true
2.3.3.5	110	true

Daten 4: metabolites_citrate_cycle_Example.txt.

nodeLabel	scientificValue	highlighted
MAL-L	50	true
OAA	60	true
CIT	100	true
ACON-C	80	true
ICIT	100	true
AKG	50	true
SUCCOA	80	true
SUCC	50	true
FUM	80	true
propionyl-CoA	2	true

Daten 5: flux_Example.txt.(als Tabelle aufgearbeitet).

reaction_id	flux	lower_ bound	upper_ bound	ec	reaction
3447	0.0000579429	0	inf	2.7.6.3	1 6-hydroxymethyl-7,8-dihydropterin + 1 ATP --> 1 6-hydroxymethyl-dihydropterin_diphosphate + 1 AMP
1170	-0.0167340762	-inf	0	2.4.2.17	1 phosphoribosyl-ATP + 1 diphosphate --> 1 ATP + 1 5-phospho-alpha-D-ribose_1-diphosphate
1170	-0.0167340762	-inf	0	2.4.2.17	1 phosphoribosyl-ATP + 1 diphosphate <-- 1 ATP + 1 5-phospho-alpha-D-ribose_1-diphosphate
1294	0.1022360815	0	inf	1.2.7.1	1 pyruvate + 1 coenzyme_A + 1 Oxidized-ferredoxin --> 1 acetyl-CoA + 1 CO2 + 1 Reduced-ferredoxin
1479	0.0319894968	0	inf	4.2.1.51	1 prephenate --> 1 phenylpyruvate + 1 H2O + 1 CO2
431	0.0452924433	0	inf	4.1.1.31	1 phosphate + 1 oxaloacetate --> 1 phosphoenolpyruvate + 1 H2O + 1 CO2
659	0.0011051773	0	inf	4.1.1.19	1 L-arginine --> 1 CO2 + 1 agmatine
5001	0.0000157882	0	inf	2.1.1.132	1 precorrin-6B + 2 S-adenosyl-L-methionine --> 1 precorrin-8x + 2 S-adenosyl-L-homocysteine + 1 CO2
306	0.1172925208	0	inf	2.2.1.6	2 pyruvate --> 1 (S)-2-acetolactate + 1 CO2
4009	-0.0001263060	-inf	inf	1.2.1.70	1 glutamate-1-semialdehyde + 1 NADP+ + 1 |GLT-tRNAs| <=> 1 |Charged-GLT-tRNAs| + 1 NADPH
5056	0.0000157882	0	inf	2.5.1.17	1 cob(I)yrinate_a,c-diamide + 1 ATP --> 1 adenosyl-cobyrinate_a,c-diamide + 1 PPPi
10427	0.0000007894	-inf	inf	2.7.8.-	2 L-1-PHOSPHATIDYL-GLYCEROL <=> 1 CARDIOLIPIN + 1 glycerol
10430	0.0000328396	-inf	inf	2.7.8.5	1 CDPDIACYLGLYCEROL + 1 sn-glycerol-3-phosphate <=> 1 CMP + 1 L-1-PHOSPHATIDYL-GLYCEROL-P + 1 H+
5054	-0.0000078941	-inf	0	1.16.8.1	2 cob(I)yrinate_a,c-diamide + 1 FMN + 3 H+ <-- 2 cob(II)yrinate_a,c-diamide + 1 FMNH2
6266	0.0000157882	0	inf	6.3.1.10	1 adenosyl-cobyrate + 1 (R)-1-amino-2-propanol_O-2-phosphate + 1 ATP --> 1 H+ + 1 adenosyl-cobinamide_phosphate + 1 ADP + 1 phosphate
5058	0.0000157882	-inf	inf	2.7.7.62	3 H+ + 1 adenosyl-cobinamide_phosphate + 1 GTP <=> 1 adenosylcobinamide-GDP + 1 diphosphate
13354	0.0001212537	0	inf	2.3.1.15	1 |Fatty-Acyl-CoA| + 1 sn-glycerol-3-phosphate --> 1 ACYL-SN-GLYCEROL-3P + 1 coenzyme_A

12966	0.0001212537	0	inf	2.7.7.41	1 CTP + 1 L-PHOSPHATIDATE + 1 H+ --> 1 CDPDIACYLGLYCEROL + 1 diphosphate
9588	0.0001212537	0	inf	2.3.1.51	1 |Fatty-Acyl-CoA| + 1 ACYL-SN-GLYCEROL-3P --> 1 L-PHOSPHATIDATE + 1 coenzyme_A
13506	0.0000884142	0	inf	4.1.1.65	1 L-1-PHOSPHATIDYL-SERINE + 1 H+ --> 1 L-1-PHOSPHATIDYL-ETHANOLAMINE + 1 CO2
13492	0.0000328396	-inf	inf	3.1.3.27	1 L-1-PHOSPHATIDYL-GLYCEROL-P + 1 H2O <=> 1 L-1-PHOSPHATIDYL-GLYCEROL + 1 phosphate
10431	0.0000884142	-inf	inf	2.7.8.8	1 CDPDIACYLGLYCEROL + 1 L-serine <=> 1 CMP + 1 L-1-PHOSPHATIDYL-SERINE + 1 H+
5392	0.0001263060	0	inf	6.1.1.17	1 |GLT-tRNAs| + 1 L-glutamate + 1 ATP --> 1 |Charged-GLT-tRNAs| + 1 diphosphate + 1 AMP
9603	0.0002425075	0	inf	2.3.1.86	1 acetyl-CoA + 1 malonyl-CoA + 2 NADH + 2 NADPH + 4 H+ --> 1 |Fatty-Acyl-CoA| + 1 coenzyme_A + 1 CO2 + 2 NAD+ + 2 NADP+
5061	0.0000157882	0	inf	6.3.5.10	1 adenosyl-cobyrinate_a,c-diamide + 4 L-glutamine + 4 ATP + 4 H2O --> 4 H+ + 4 L-glutamate + 1 adenosyl-cobyrate + 4 ADP + 4 phosphate
6982	0.0000509960	0	inf		1 5-amino-6-(5-phospho-D-ribitylamino)uracil + 1 H2O --> 1 5-amino-6-(D-ribitylamino)uracil + 1 phosphate
6268	0.0000157882	0	inf		1 L-threonine + 1 ATP --> 1 H+ + 1 L-threonine_O-3-phosphate + 1 ADP
1738	0.0623554339	0	inf		1 (2S)-2-isopropyl-3-oxosuccinate --> 1 4-methyl-2-oxopentanoate + 1 CO2
H2O_trans	-90980085926	-inf	inf		1 H2O_ex <=> 1 H2O
H2O_flux	-90980085926	-inf	inf		<=> 1 H2O_ex
oxygen_trans	53387811897	0	inf		1 oxygen_ex <=> 1 oxygen
oxygen_flux	53387811897	-inf	inf		<=> 1 oxygen_ex
phosphate_trans	0.1612930447	-inf	inf		1 phosphate_ex <=> 1 phosphate
phosphate_flux	0.1612930447	-inf	inf		<=> 1 phosphate_ex
ammonia_trans	13458089521	0	inf		1 ammonia_ex --> 1 ammonia

ammonia_flux	13458089521	-inf	inf		<=> 1 ammonia_ex
sulfate_trans	0.0295701552	0	inf		1 sulfate_ex --> 1 sulfate
sulfate_flux	0.0295701552	-inf	inf		<=> 1 sulfate_ex
Co2+_trans	0.0000157882	0	inf		1 Co2+_ex --> 1 Co2+
Co2+_flux	0.0000157882	-inf	inf		<=> 1 Co2+_ex
CO2_trans	-48282617380	-inf	0		1 CO2_ex <-- 1 CO2
CO2_flux	-48282617380	-inf	inf		<=> 1 CO2_ex
H+_trans	-1,20445E+11	-inf	0		1 H+_ex <-- 1 H+
H+_flux	-1,20445E+11	-inf	inf		<=> 1 H+_ex
4,5-dihydroxy-2,3-pentanedione_trans	-0.0001683027	-inf	0		1 4,5-dihydroxy-2,3-pentanedione_ex <-- 1 4,5-dihydroxy-2,3-pentanedione
4,5-dihydroxy-2,3-pentanedione_flux	-0.0001683027	-inf	inf		<=> 1 4,5-dihydroxy-2,3-pentanedione_ex
2196	1,02443E+11	-inf	inf	1.6.5.-	1 |Ubiquinones| + 1 NADH --> 1 |Ubiquinols| + 1 NAD+
2196	1,02443E+11	-inf	inf	1.6.5.3	1 NADH + 1 |Ubiquinones| + 5 H+ <=> 1 NAD+ + 1 |Ubiquinols| + 4 H+_CCO-PERI-BAC
2197	0.4310350335	-inf	inf	1.3.5.1	1 |Ubiquinones| + 1 succinate <=> 1 fumarate + 1 |Ubiquinols|
14348	1,06753E+11	0	inf	1.10.2.2	2 |Cytochromes-C-Oxidized| + 1 |Ubiquinols| + 2 H+ --> 2 |Cytochromes-C-Reduced| + 1 |Ubiquinones| + 4 H+_CCO-PERI-BAC
13955	53376444359	0	inf	1.9.3.1	1 oxygen + 4 |Cytochromes-C-Reduced| + 8 H+ --> 2 H2O + 4 |Cytochromes-C-Oxidized| + 4 H+_CCO-PERI-BAC
79	-2,62572E+11	-inf	inf	3.6.4.13	1 ATP + 1 H2O --> 1 ADP + 1 phosphate
79	-2,62572E+11	-inf	inf	3.6.4.6	1 H2O + 1 ATP --> 1 phosphate + 1 ADP
79	-2,62572E+11	-inf	inf	3.6.4.10	1 ATP + 1 H2O --> 1 ADP + 1 phosphate

79	-2,62572E+11	-inf	inf	3.6.4.12	1 ATP + 1 H2O --> 1 ADP + 1 phosphate
79	-2,62572E+11	-inf	inf	3.6.1.3	1 H2O + 1 ATP --> 1 phosphate + 1 ADP
79	-2,62572E+11	-inf	inf	3.6.3.51	1 H2O + 1 ATP --> 1 phosphate + 1 ADP
79	-2,62572E+11	-inf	inf	3.6.4.3	1 H2O + 1 ATP --> 1 phosphate + 1 ADP
79	-2,62572E+11	-inf	inf	3.6.3.6	1 H+ + 1 H2O + 1 ATP --> 1 H+_CCO-EXTRACELLULAR + 1 phosphate + 1 ADP
79	-2,62572E+11	-inf	inf	3.6.3.27	1 ATP + 1 H2O + 1 phosphate_CCO-EXTRACELLULAR --> 1 ADP + 2 phosphate
79	-2,62572E+11	-inf	inf	5.4.99.-	1 (S)-2,3-epoxysqualene --> 1 lupeol
79	-2,62572E+11	-inf	inf	3.6.3.14	4 H+ + 1 H2O + 1 ATP <=> 4 H+_CCO-PERI-BAC + 1 phosphate + 1 ADP
unknown-flux	0.0000157882	0	inf		1 UNKNOWN -->
13252	0.0000157882	0	inf	1.14.99.40	1 FMNH2 + 1 oxygen + 1 NADH + 1 H+ --> 1 5,6-dimethylbenzimidazole + 1 D-erythrose-4-phosphate + 1 UNKNOWN + 1 NAD+
8296	0.0011051773	-inf	inf	4.3.2.1	1 canavaninosuccinate --> 1 L-canavanine + 1 fumarate
8296	0.0011051773	-inf	inf	2.6.1.5	1 2-oxo-4-methylthiobutanoate + 1 L-alanine <=> 1 L-methionine + 1 pyruvate
13330	17000000000	0	inf		1 phosphoenolpyruvate + 1 beta-D-glucose_CCO-PERI-BAC --> 1 beta-D-glucose-6-phosphate + 1 pyruvate
Transp_C_trans_beta-D-glucose_CCO-PERI-BAC	17000000000	0	1,7E+10		1 beta-D-glucose_CCO-PERI-BAC_ex --> 1 beta-D-glucose_CCO-PERI-BAC
Transp_C_flux_beta-D-glucose_CCO-PERI-BAC	17000000000	-inf	inf		<=> 1 beta-D-glucose_CCO-PERI-BAC_ex
ATP_maintenance_requirement	90000000000	9E+10	9E+10		1 ATP + 1 H2O --> 1 ADP + 1 H+ + 1 phosphate
GFP-reaction	0.0003828921	0.0003829	inf		10 L-alanine + 2 L-cysteine + 16 L-glutamate + 18 L-aspartate + 22 glycine + 11 L-phenylalanine + 13 L-isoleucine + 10 L-histidine + 21 L-lysine + 5 L-methionine + 20 L-leucine + 13 L-asparagine + 8 L-glutamine + 10 L-proline + 13 L-serine + 7 L-

				arginine + 18 L-threonine + 2 L-tryptophan + 17 L-valine + 11 L-tyrosine + 1.96505 ATP --> 1.96505 ADP + 1.96505 phosphate + 1.96505 H+ + 1 GFP
Biomass-reaction	0.1578824677	0	inf	0.000367 10-formyl-tetrahydrofolate + 0.266902 L-alanine + 0.00467 AMP + 0.193021 L-arginine + 0.147987 L-asparagine + 0.148014 L-aspartate + 105.053 ATP + 5e-06 CARDIOLIPIN + 0.000251 CDP + 0.001042 CMP + 0.038902 CTP + 0.05699 L-cysteine + 0.022982 dATP + 0.01738 dCTP + 0.017398 dGTP + 0.022899 dTTP + 0.00018 GDP + 0.260335 L-glutamine + 0.260378 L-glutamate + 0.408288 glycine + 0.000503 GMP + 0.00779 prenyl-P-P-GlcNAc-ManNAc-P-Gro + 0.062667 GTP + 105 H2O + 0.081739 L-histidine + 0.269905 L-isoleucine + 0.346445 L-leucine + 0.323115 L-lysine + 0.000198 L-1-PHOSPHATIDYL-GLYCEROL + 0.113326 L-methionine + 0.000266 menaquinol-7 + 0.016164 NAD+ + 0.000934 NADP+ + 0.000216 NADPH + 0.0509085 a_peptidoglycan_dimer_(meso-diaminopimelate_containing) + 0.175939 L-phenylalanine + 0.000918 diphosphate + 0.160642 L-proline + 0.00056 L-1-PHOSPHATIDYL-ETHANOLAMINE + 0.216213 L-serine + 0.186317 L-threonine + 0.054336 L-tryptophan + 0.110824 L-tyrosine + 0.041501 UTP + 0.306734 L-valine + 0.000223 FAD + 0.007 spermidine + 0.000223 pyridoxal_5'-phosphate + 0.0001 coenzyme_B12 --> 104.997 ADP + 105 H+ + 104.986 phosphate + 1 Biomass_megaterium
GFP_exporter	0.0003828921	0	inf	1 GFP -->
Biomass	0.1578824677	0	1E+16	1 Biomass_megaterium -->

Klasse *NetworkGraphicsWidget*

Tabelle A.2: Übersicht der Methoden von *NetworkGraphicsWidget* und die skriptbaren Funktionen.

Rüchgabe	Methode	Attribut (*QProperty*)	Beschreibung
virtual QString	label () const	label	Gibt das gesetzte Label zurück. Das Label ist der Text der gerendert wird.
virtual void	setLabel (const QString &label)	label	Setzt für das Label "label".
virtual bool	paintLabel ()		Gibt zurück, ob das Label gerendert werden soll.
virtual void	setPaintLabel (bool paintLabel)		Wenn "paintLabel" wahr ist, wird das Label gerendert, andernfalls nicht.
virtual QString	recommendedName () const	recommendedName	Gibt den gesetzten Gebrauchsnamen zurück.
virtual void	setRecommendedName (const QString &name)	recommendedName	Setzt für den Gebrauchsnamen "name".
virtual QString	scriptObjectName () const	scriptObjectName	Gibt den gesetzten „scriptObjectName zurück. Der scriptObjectName ist der Name der in einem Script zu verwenden ist.
virtual void	setScriptObjectName (const QString &name)	scriptObjectName	Setzt für scriptObjectName "name".
virtual QString	htmlLink () const	htmlLink	Gibt den gesetzten Html-Link zurück.
virtual void	setHtmlLink (const QString &link)	htmlLink	Setzt für das Html-Link "link".
virtual QString	curveData ()	curveData	Gibt die gesetzten Daten der Kurve zurück.
virtual void	setCurveData (const QString &curveData)	curveData	Setzt für die Daten der Kurve "curveData".
virtual ScientificPlotCurve *	scientificPlotCurve ()		Gibt den gesetzten Pointer auf eine ScientificPlotCurve zurück.
virtual void	setScientificPlotCurve (ScientificPlotCurve *curve)		Setzt für den Pointer auf eine ScientificPlotCurve "curve".
virtual NetworkObject *	subNetworkObject ()		Gibt den gesetzten Pointer auf ein NetworkObject zurück.
virtual void	setSubNetworkObject (NetworkObject *networkObject)		Setzt für den Pointer auf ein NetworkObject "networkObject ".
virtual QString	getInformation ()		Gibt Informationen zu dem NetzwerkGraphicsWidget zurück.

virtual void	updateToolTip ()		Dient als Update-Funktion. Muss nach Änderung der Anzeige von Tooltip-Funktionen aufgerufen werden.
QStringList	synonyms ()	synonyms	Gibt die gesetzten Synonyme als Liste zurück.
void	addSynonym (QString synonym)		Fügt zu der Liste der Synonyme das Synonym "synonym" zu.
void	addSynonyms (QStringList synonyms)	synonyms	Fügt zu der Liste der Synonyme die Liste der Synonyme "synonyms" zu.
void	removeSynonym (QString synonym)		Entfernt aus der Liste der Synonyme das Synonym "synonym".
void	removeSynonyms (QStringList synonyms)		Entfernt aus der Liste der Synonyme die Liste der Synonyme "synonyms".
void	clearSynonyms ()		Löscht alle gesetzten Synonyme.
QPixmap	pixmap () const	pixmap	Gibt das gesetzte Pixmap zurück.
void	setPixmap (const QPixmap &pixmap)	pixmap	Setzt für das Pixmap "pixmap".
Qt::TransformationMode	transformationMode () const	transformationMode	Gibt den Enumerator-Wert "Qt::TransformationMode" des NetzwerkGraphicsWidget zurück.
void	setTransformationMode (Qt::TransformationMode mode)	transformationMode	Setzt für den Enumerator-Wert "Qt::TransformationMode" des NetzwerkGraphicsWidgets "mode".
QPointF	offset () const	offset	Gibt den Wert der Verschiebung als 2d Koordinatenpunkt zurück.
void	setOffset (const QPointF &offset)	offset	Setzt für den Wert der Verschiebung als 2d Koordinatenpunkt "offset".
void	setOffset (qreal x, qreal y)		Setzt für den Wert der Verschiebung als x und y Wert "x" und "y".
virtual bool	isSelected () const	selected	Gibt zurück, ob das NetworkGrapcisWidget selektiert ist.
virtual void	setSelected (bool selected)	selected	Wenn "selected" wahr ist, wird das NetworkGrapcisWidget selektiert, andernfalls nicht.
virtual void	toggleSelection ()		Wenn das NetworkGrapcisWidget selektiert ist wird es deselektiert, sonst selektiert.
Qt::TextElideMode	textElideMode () const	textElideMode	Gibt den Enumerator-Wert "Qt::TextElideMode " des NetzwerkGraphicsWidget zurück.
void	setTextElideMode (const Qt::TextElideMode &textElideMode)	textElideMode	Setzt für den Enumerator-Wert "Qt::TextElideMode " des NetzwerkGraphicsWidgets "textElideMode".
DataMode	dataMode () const	dataMode	Gibt den Enumerator-Wert "DataMode" des NetzwerkGraphicsWidget zurück.

void	setDataMode (const DataMode &mode)	dataMode	Setzt für den Enumerator-Wert "DataMode" des NetzwerkGraphicsWidgets "dataMode".
QPen	pen () const	pen	Gibt den Stift der Rahmenlinie zurück.
void	setPen (const QPen &pen)	pen	Setzt für den Stift der Rahmenlinie "pen".
QBrush	penBrush () const	penBrush	Gibt den Pinsel der Rahmenlinie zurück.
void	setPenBrush (const QBrush &brush)	penBrush	Setzt für den Pinsel der Rahmenlinie "penBrush".
QColor	penColor () const	penColor	Gibt die Farbe des Stifts der Rahmenlinie zurück.
void	setPenColor (const QColor &color)	penColor	Setzt für die Farbe des Stifts der Rahmenlinie "penColor".
Qt::BrushStyle	penPattern () const	penPattern	Gibt den Enumerator-Wert "Qt::BrushStyle " für das Muster des Stifts der Rahmenlinie zurück.
void	setPenPattern (const Qt::BrushStyle &pattern)	penPattern	Setzt für den Enumerator-Wert "Qt::BrushStyle " für das Muster des Stifts der Rahmenlinie "penPattern".
QBrush	fillBrush () const	fillBrush	Gibt den Pinsel der Füllfläche zurück.
void	setFillBrush (const QBrush &brush)	fillBrush	Setzt für den Pinsel der Füllfläche "fillBrush".
QColor	fillColor () const	fillColor	Gibt die Farbe des Pinsels der Füllfläche zurück.
void	setFillColor (const QColor &color)	fillColor	Setzt für die Farbe des Pinsels der Füllfläche "fillColor".
Qt::BrushStyle	fillPattern () const	fillPattern	Gibt den Enumerator-Wert "Qt::BrushStyle " für das Muster der Füllfläche zurück.
void	setFillPattern (const Qt::BrushStyle &pattern)	fillPattern	Setzt für den Enumerator-Wert "Qt::BrushStyle " für das Muster der Füllfläche "fillPattern".
QFont	labelFont () const	labelFont	Gibt die Schriftart des gerenderten Labels zurück.
void	setLabelFont (const QFont &labelFont)	labelFont	Setzt für die Schriftart des gerenderten Labels "labelFont".
bool	tagTopLeft () const	tagTopLeft	Gibt zurück, ob das ein kleines Etikett oben links innerhalb des NetworkGrapcisWidgets gerendert wird.
void	setTagTopLeft (bool tag)	tagTopLeft	Wenn "tagTopLeft" wahr ist, wird ein kleines Etikett oben links innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	tagTopRight () const	tagTopRight	Gibt zurück, ob das ein kleines Etikett oben rechts innerhalb des NetworkGrapcisWidgets gerendert wird.

void	setTagTopRight (bool tag)	tagTopRight	Wenn "tagTopRight" wahr ist, wird ein kleines Etikett oben rechts innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	tagCenterLeft () const	tagCenterLeft	Gibt zurück, ob das ein kleines Etikett in der Mitte links innerhalb des NetworkGrapcisWidgets gerendert wird.
void	setTagCenterLeft (bool tag)	tagCenterLeft	Wenn "tagCenterLeft" wahr ist, wird ein kleines Etikett in der Mitte links innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	tagCenterRight () const	tagCenterRight	Gibt zurück, ob das ein kleines Etikett in der Mitte rechts innerhalb des NetworkGrapcisWidgets gerendert wird.
void	setTagCenterRight (bool tag)	tagCenterRight	Wenn "tagCenterRight" wahr ist, wird ein kleines Etikett in der Mitte rechts innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	tagBottomLeft () const	tagBottomLeft	Gibt zurück, ob das ein kleines Etikett unten links innerhalb des NetworkGrapcisWidgets gerendert wird.
void	setTagBottomLeft (bool tag)	tagBottomLeft	Wenn "tagBottomLeft" wahr ist, wird ein kleines Etikett unten links innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	tagBottomRight () const	tagBottomRight	Gibt zurück, ob das ein kleines Etikett unten rechts innerhalb des NetworkGrapcisWidgets gerendert wird.
void	setTagBottomRight (bool tag)	tagBottomRight	Wenn "tagBottomRight" wahr ist, wird ein kleines Etikett unten rechts innerhalb des NetworkGrapcisWidgets gerendert, andernfalls nicht.
bool	showProxyWidget () const	showProxyWidget	Gibt zurück, ob das gesetzte ProxyWidget angezeigt wird.
void	setShowProxyWidget (bool enable)	showProxyWidget	Wenn "showProxyWidget" wahr ist, wird das gesetzte ProxyWidget angezeigt, andernfalls nicht.
bool	showArrowGraphicsWidget () const	showArrowGraphicsWidget	Gibt zurück, ob das ArrowGraphicsWidget angezeigt wird.
void	setShowArrowGraphicsWidget (bool enable)	showArrowGraphicsWidget	Wenn "showArrowGraphicsWidget" wahr ist, wird das ArrowGraphicsWidget angezeigt, andernfalls nicht.
bool	showStatusGraphicsWidget () const	showStatusGraphicsWidget	Gibt zurück, ob das StatusGraphicsWidget angezeigt wird.
void	setShowStatusGraphicsWidget (bool enable)	showStatusGraphicsWidget	Wenn "showStatusGraphicsWidget" wahr ist, wird das StatusGraphicsWidget angezeigt, andernfalls nicht.
bool	enableGraphicsBlurEffect () const	enableGraphicsBlurEffect	Gibt zurück, ob ein Blur-Effekt auf das NetworkGrapcisWidgets angewendet wird.
void	setGraphicsBlurEffectEnable (bool enable)	enableGraphicsBlurEffect	Wenn "enableGraphicsBlurEffect" wahr ist, wird ein Blur-Effekt auf das NetworkGrapcisWidgets angewendet, andernfalls nicht.
bool	enableGraphicsColorizeEffect () const	enableGraphicsColorizeEffect	Gibt zurück, ob ein Farb-Effekt auf das NetworkGrapcisWidgets angewendet wird.
void	setGraphicsColorizeEffectEnable (bool enable)	enableGraphicsColorizeEffect	Wenn "enableGraphicsColorizeEffect" wahr ist, wird ein Farb-Effekt auf das NetworkGrapcisWidgets angewendet, andernfalls nicht.
bool	enableGraphicsDropShadowEffect () const	enableGraphicsDropShadowEffect	Gibt zurück, ob ein Schlagschatten-Effekt auf das NetworkGrapcisWidgets angewendet wird.

void	setGraphicsDropShadowEffectEnable (bool enable)	enableGraphicsDropShadowEffect	Wenn "enableGraphicsDropShadowEffect" wahr ist, wird ein Schlagschatten-Effekt auf das NetworkGrapcisWidgets angewendet, andernfalls nicht.
bool	enableGraphicsOpacityEffect () const	enableGraphicsOpacityEffect	Gibt zurück, ob ein Transparenz-Effekt auf das NetworkGrapcisWidgets angewendet wird.
void	setGraphicsOpacityEffectEnable (bool enable)	enableGraphicsOpacityEffect	Wenn "enableGraphicsOpacityEffect" wahr ist, wird ein Transparenz Effekt auf das NetworkGrapcisWidgets angewendet, andernfalls nicht.
bool	highlighted () const	highlighted	Gibt zurück, ob das NetworkGrapcisWidgets unterlegt hervorgehoben wird.
void	setHighlighted (const bool &highlighted)	highlighted	Wenn "showPixmapToolTip" wahr ist, wird das NetworkGrapcisWidgets unterlegt hervorgehoben angezeigt, andernfalls nicht.
QBrush	highlightBrush () const	highlightBrush	Gibt den Pinsel der grafischen Hervorhebung/Unterlegung zurück.
void	setHighlightBrush (const QBrush &brush)	highlightBrush	Setzt für den Pinsel der grafischen Hervorhebung/Unterlegung "highlightBrush".
int	detailLevel () const	detailLevel	Gibt das DetailLevel zurück.
void	setDetailLevel (const int &detailLevel)	detailLevel	Setzt für das DetailLevel "detailLevel".
bool	enableTextBackgroundBox () const	enableTextBackgroundBox	Gibt zurück, ob das Label in einem kleinen Kasten angezeigt wird.
void	setEnableTextBackgroundBox (const bool &textBackgroundBox)	enableTextBackgroundBox	Wenn "enableTextBackgroundBox" wahr ist, wird das Label in einem kleinen Kasten angezeigt, andernfalls nicht.
bool	hasCollisionDetection () const	collisionDetection	Gibt zurück, ob die Kollisionsdetektion das NetworkGrapcisWidgets aktiviert ist.
void	toggleCollisionDetection (bool enable)	collisionDetection	Wenn "collisionDetection" wahr ist, ist die Kollisionsdetektion das NetworkGrapcisWidgets aktiviert, andernfalls nicht.
bool	showPixmap () const	showPixmap	Gibt zurück, ob das gesetzte Pixmap angezeigt wird.
void	setShowPixmap (bool enable)	showPixmap	Wenn "showPixmap" wahr ist, wird das gesetzte Pixmap angezeigt, andernfalls nicht.
bool	showCurveData () const	showCurveData	Gibt zurück, ob eine DatenKurve gerendert wird. Dieses ist noch nicht funktional.
void	setShowCurveData (bool enable)	showCurveData	Wenn "showCurveData" wahr ist, wird eine DatenKurve gerendert, andernfalls nicht. Dieses ist noch nicht funktional.
bool	showToolTipAsLabel () const	showToolTipAsLabel	Gibt zurück, ob der Tooltip als Labeltext angezeigt wird.
void	setShowToolTipAsLabel (bool enable)	showToolTipAsLabel	Wenn "showToolTipAsLabel" wahr ist, wird der Tooltip als Labeltext angezeigt, andernfalls nicht.
bool	showLabelToolTip () const	showLabelToolTip	Gibt zurück, ob das gesetzte Label im Tooltip angezeigt wird.

void	setShowLabelToolTip (bool enable)	showLabelToolTip	Wenn "showLabelToolTip" wahr ist, wird das gesetzte Label im Tooltip angezeigt, andernfalls nicht.
bool	showRecomendedNameToolTip () const	showRecomendedNameToolTip	Gibt zurück, ob der gesetzte Gebrauchsname im Tooltip angezeigt wird.
void	setShowRecomendedNameToolTip (bool enable)	showRecomendedNameToolTip	Wenn "showRecomendedNameToolTip" wahr ist, wird der gesetzte Gebrauchsname im Tooltip angezeigt, andernfalls nicht.
bool	showSynonymsToolTip () const	showSynonymsToolTip	Gibt zurück, ob alle gesetzten Synonyme im Tooltip angezeigt werden.
void	setShowSynonymsToolTip (bool enable)	showSynonymsToolTip	Wenn "showPixmapToolTip" wahr ist, werden alle gesetzten Synonyme im Tooltip angezeigt, andernfalls nicht.
bool	showScriptObjectNameToolTip () const	showScriptObjectNameToolTip	Gibt zurück, ob der Name des Objektes gesetzt für die Skript-Engine im Tooltip angezeigt wird.
void	setShowScriptObjectNameToolTip (bool enable)	showScriptObjectNameToolTip	Wenn "showScriptObjectNameToolTip" wahr ist, wird der Name des Objektes gesetzt für die Skript-Engine im Tooltip angezeigt, andernfalls nicht.
bool	showHtmlLinkToolTip () const	showHtmlLinkToolTip	Gibt zurück, ob der Html Link für das NetworkGrapcisWidgets im Tooltip angezeigt wird.
void	setShowHtmlLinkToolTip (bool enable)	showHtmlLinkToolTip	Wenn "showHtmlLinkToolTip" wahr ist, wird der Html Link für das NetworkGrapcisWidgets im Tooltip angezeigt, andernfalls nicht.
bool	showCurveDataToolTip () const	showCurveDataToolTip	Gibt zurück, ob eine DatenKurve im Tooltip gerendert wird. Dieses ist noch nicht funktional.
void	setShowCurveDataToolTip (bool enable)	showCurveDataToolTip	Wenn "showCurveData" wahr ist, wird eine DatenKurve im Tooltip gerendert, andernfalls nicht. Dieses ist noch nicht funktional.
bool	showPixmapToolTip () const	showPixmapToolTip	Gibt zurück, ob das gesetzte Pixmap im Tooltip angezeigt wird.
void	setShowPixmapToolTip (bool enable)	showPixmapToolTip	Wenn "showPixmapToolTip" wahr ist, wird das gesetzte Pixmap im Tooltip angezeigt, andernfalls nicht.
virtual NetworkObjectStyle	networkObjectStyle ()		Gibt das aktuelle NetworkObjectStyle des NetworkGrapcisWidgets zurück.
static NetworkObjectStyle	defaultNetworkObjectStyle ()		Gibt das NetworkObjectStyle der Vorgabe des NetworkGrapcisWidgets zurück.
virtual void	resetToDefault ()	resetToDefault	Setzt das NetworkGrapcisWidgets auf die Ursprungswerte zurück.
void	setPenColor (int red, int green, int blue, int alpha)	setPenColor	Setzt für die Farbe des Stifts der Rahmenlinie die RGBA Werte "red", "green", "blue" und "alpha".
void	setFillColor (int red, int green, int blue, int alpha)	setFillColor	Setzt für die Farbe des Pinsels der Füllfläche die RGBA Werte "red", "green", "blue" und "alpha".

Klasse *NodeObject*

Tabelle A.3: Übersicht der Methoden von *NodeObject* und die skriptbaren Funktionen.

Rüchgabe	Methode	Attribut (*QProperty*)	Beschreibung
int	nodeId () const	nodeId	Gibt die Knoten-ID zurück.
void	setNodeId (const int &nodeId)	nodeId	Setzt für die Knoten-ID "nodeID".
int	oldNodeId () const	oldNodeId	Gibt die alte Knoten-ID zurück.
void	setOldNodeId (const int &oldNodeId)	oldNodeId	Setzt für die alte Knoten-ID "oldNodeId".
QString	nodeName () const	nodeName	Gibt den Knoten-Namen zurück.
void	setNodeName (const QString &nodeName)	nodeName	Setzt für den Knoten-Namen "nodeName".
QString	nodeNameType () const	nodeNameType	Gibt den Typ des Knoten-Namens zurück. Dieses ist eine Art Gruppierung.
void	setNodeNameType (const QString &nodeNameType)	nodeNameType	Setzt für den Typ des Knoten-Namens "nodeNameType". Dieses ist eine Art Gruppierung.
QString	nodeUsedName () const	nodeUsedName	Gibt den Gebrauchsnamen des Knoten zurück.
void	setNodeUsedName (const QString &usedName)	nodeUsedName	Setzt für den Gebrauchsnamen des Knoten "nodeUsedName".
double	height () const	height	Gibt die Höhe des Knoten zurück.
void	setHeight (const double &height)	height	Setzt für die Höhe des Knoten "height".
double	width () const	width	Gibt die Breite des Knoten zurück.
void	setWidth (const double &width)	width	Setzt für die Breite des Knoten "width".
const char *	oldPropertyName () const		Gibt den Namen des alten Attributs des Knoten zurück. Dient für die Zwischenspeicherung.
void	setOldPropertyName (const char *oldPropertyName)		Setzt für den Namen des alten Attributs des Knoten "oldPropertyName". Dient für die Zwischenspeicherung.
QVariant	oldPropertyValue () const	oldPropertyValue	Gibt den Wert des alten Attributs des Knoten zurück. Dient für die Zwischenspeicherung.

void	setOldPropertyValue (QVariant &oldPropertyValue)	oldPropertyValue	Setzt für den Wert des alten Attributs des Knoten "oldPropertyValue". Dient für die Zwischenspeicherung.
QString	analysedProperty ()	analysedProperty	Gibt den Namen des analysierten Attributs des Knoten zurück.
void	setAnalysedProperty (QString property)	analysedProperty	Setzt für den Namen des analysierten Attributs des Knoten "property".
PaintType	paintType () const	paintType	Gibt den Enumerator-Wert "PaintType" des Knoten zurück.
void	setPaintType (const PaintType &paintType)	paintType	Setzt für den Enumerator-Wert "PaintType" des Knoten "paintType".
NodeType	nodeType () const	nodeType	Gibt den Enumerator-Wert "NodeType" des Knoten zurück.
void	setNodeType (const NodeObject::NodeType &nodeType)	nodeType	Setzt für den Enumerator-Wert "NodeType" des Knoten "nodeType".
QString	nodeTypeName () const	nodeTypeName	Gibt den Namen des Typen (geometrische Form) des Knoten zurück.
void	setNodeTypeName (const QString &nodeType)	nodeTypeName	Setzt für den Name des Typen (geometrische Form) des Knoten "nodeNameType".
NodeGradientType	nodeGradientType () const	nodeGradientType	Gibt den Enumerator-Wert "NodeGradientType" des Knoten zurück.
void	setNodeGradientType (const NodeObject::NodeGradientType &nodeGradientType)	nodeGradientType	Setzt für den Enumerator-Wert "NodeGradientType" des Knoten "nodeGradientType".
double	scientificValue ()	scientificValue	Gibt den wissenschaftlichen Wert des Knoten zurück. Wird für die Datenabbildung in BRIME benötigt.
void	setScientificValue (const double &value)	scientificValue	Setzt für den wissenschaftlichen Wert des Knoten "scientificValue". Wird für die Datenabbildung in BRIME benötigt.
double	deltaX ()	deltaX	Gibt delta X des Knoten zurück. Wird für das Kräfte-basierte Layout verwendet.
void	setDeltaX (const double &value)	deltaX	Setzt für delta X des Knoten "deltaX". Wird für das Kräfte-basierte Layout verwendet.
double	deltaY ()	deltaY	Gibt delta Y des Knoten zurück. Wird für das Kräfte-basierte Layout verwendet.
void	setDeltaY (const double &value)	deltaY	Setzt für delta Y des Knoten "deltaY". Wird für das Kräfte-basierte Layout verwendet.
double	mass ()	mass	Gibt die Masse des Knoten zurück. Wird für das Kräfte-basierte Layout verwendet.
void	setMass (const double &value)	mass	Setzt für die Masse des Knoten "mass". Wird für das Kräfte-basierte Layout verwendet.

double	edgesFluxValue ()	edgesFluxValue	Gibt den Fluss Wert der Kanten des Knoten zurück. Wird für das Kräfte-basierte Layout verwendet.
void	setEdgesFluxValue (const double &value)	edgesFluxValue	Setzt für den Fluss Wert der Kante des Knoten "edgesFluxValue".
QImage	nodeImage () const	nodeImage	Gibt das gesetzte Knoten-Bild des Knoten zurück.
void	setNodeImage (const QImage &image)	nodeImage	Setzt für das Knoten-Bild des Knoten "nodeImage".
bool	visited () const	visited	Gibt zurück, ob der Knoten schon besucht wurde oder nicht.
void	setVisited (const bool &visited)	visited	Wenn "visited" wahr ist, wird der Knoten als besucht markiert, andernfalls nicht.
bool	available () const	available	Gibt zurück, ob der Knoten verfügbar ist oder nicht.
void	setAvailable (const bool &available)	available	Wenn "available" wahr ist, wird der Knoten als verfügbar markiert, andernfalls nicht.
bool	tagFixedAnchor () const	tagFixedAnchor	Gibt zurück, ob der Knoten für Layouting als fixiert angesehen wird oder nicht.
void	setTagFixedAnchor (const bool &tagFixedAnchor)	tagFixedAnchor	Wenn "tagFixedAnchor" wahr ist, wird der Knoten für Layouting als fixiert angesehen, andernfalls nicht.
void	save ()		Speichert den aktuellen Zustand des Knoten im Zwischenspeicher.
void	restore ()		Ruft den gespeicherten Zustand des Knoten aus dem Zwischenspeicher ab.
int	branchingCount ()	branchingCount	Gibt die Anzahl benachbarter Knoten zurück.
int	branchingCount (AbstractNetworkContainer *networkContainer)		Gibt die Anzahl benachbarter Knoten innerhalb des übergebenen Teilnetzwerk "networkContainer" zurück.
bool	isHubNode ()	isHubNode	Gibt zurück, ob der Knoten einen Knotengrad größer 2 besitzt.
bool	isDeadEndNode ()	isDeadEndNode	Gibt zurück, ob der Knoten einen Knotengrad gleich 1 besitzt.
bool	isLinearWayNode ()	isLinearWayNode	Gibt zurück, ob der Knoten einen Knotengrad gleich 2 besitzt.
bool	isNotConnectedNode ()	isNotConnectedNode	Gibt zurück, ob der Knoten einen Knotengrad gleich 0 besitzt.
bool	isConnected (NodeObject *node)		Gibt zurück, ob der Knoten mit dem übergebenen Knoten durch einen Kante verbunden ist.
bool	isConnected (NodeObject *node, AbstractNetworkContainer		Gibt zurück, ob der Knoten mit dem übergebenen Knoten durch einen Kante innerhalb des übergebenen Teilnetzwerks "networkContainer" verbunden ist.

	*networkContainer)		
QList< EdgeObject * >	getEdges ()	getEdges	Gibt eine Liste mit Pointern auf die Kanten zurück, die dem Knoten zugeordnet sind.
QList< EdgeObject * >	getEdges (AbstractNetworkContainer *networkContainer)		Gibt eine Liste mit Pointern auf die Kanten zurück, die dem Knoten innerhalb des übergebenen Teilnetzwerks "networkContainer" zugeordnet sind.
QList< GroupObject * >	getGroups ()	getGroups	Gibt eine Liste mit Pointern auf die Gruppen zurück, die dem Knoten zugeordnet sind.
NodeObject *	getNeighbourNode (EdgeObject *edge)		Gibt den Knoten zurück, sofern vorhanden, zwischen diesem und der übergebenen Kante "edge", sonst 0.
QList< NodeObject * >	getNeighbourNodes ()	getNeighbourNodes	Gibt eine Liste mit Pointern auf die Nachbarknoten zurück.
QList< NodeObject * >	getNeighbourNodes (AbstractNetworkContainer *networkContainer)		Gibt eine Liste mit Pointern auf die Nachbarknoten innerhalb des übergebenen Teilnetzwerks "networkContainer" zurück.
QList< NodeObject * >	getNeighbourNodesIfHasProperty (const char *compareProperty, QVariant compareValue)		Gibt eine Liste mit Pointern auf die Nachbarknoten zurück, sofern die Nachbarknoten den Wert "compareValue" für das übergebene Attribut "compareProperty" besitzen.
QSet< NodeObject * >	getNeighbourNodesSet ()	getNeighbourNodesSet	Gibt ein Set mit Pointern auf die Knoten zurück, die dem Knoten zugeordnet sind.
QSet< EdgeObject * >	getNeighbourEdgesSet ()	getNeighbourEdgesSet	Gibt ein Set mit Pointern auf die Kanten zurück, die dem Knoten zugeordnet sind.
EdgeObject *	getEdge (NodeObject *otherNode)		Gib die Kante zurück, sofern vorhanden, zwischen diesem und dem übergebenen Knoten "otherNode", sonst 0.
EdgeObject *	getEdge (NodeObject *otherNode, AbstractNetworkContainer *networkContainer)		Gib die Kante zurück, sofern vorhanden, zwischen diesem und dem übergebenen Knoten "otherNode" innerhalb des übergebenen Teilnetzwerk "networkContainer", sonst 0.
NodeObject *	nextNeighbour (NodeObject *firstNeighbour)		Gibt die nächsten benachbarten Knoten zurück.

Klasse *EdgeObject*

Tabelle A.4: Übersicht der Methoden von *EdgeObject* und die skriptbaren Funktionen.

Rüchgabe	Methode	Attribut (*QProperty*)	Beschreibung
int	edgeId () const	edgeId	Gibt die Kanten-ID zurück.
void	setEdgeId (const int &edgeId)	edgeId	Setzt für die Kanten-ID "groupID".
QString	edgeName () const	edgeName	Gibt die Kanten-Name zurück.
void	setEdgeName (const QString &edgeName)	edgeName	Setzt für die Kanten-Name "groupID".
qreal	angle () const	angle	Gibt den Winkel der Kante zurück.
void	setAngle (qreal angle)	angle	Setzt für den Winkel der Kante "angle".
qreal	length () const	length	Gibt die Länge der Kante zurück.
void	setLength (qreal length)	length	Setzt für die Länge der Kante "length".
double	weight ()	weight	Gibt das Gewicht der Kante zurück.
void	setWeight (const double &value)	weight	Setzt für das Gewicht der Kante "weight".
double	thickness ()	thickness	Gibt die Strichstärke der Kante zurück.
void	setThickness (const double &value)	thickness	Setzt für die Strichstärke der Kante "thickness".
bool	isDirected ()	isDirected	Gibt zurück, ob die Kante gerichtet ist oder nicht.
void	setDirected (bool directed)	isDirected	Wenn "isDirected" wahr ist, ist die Kante gerichtet, andernfalls nicht.
double	edgeHeadLeft () const	edgeHeadLeft	Gibt die Größe des Pfeils (links) der Kante zurück.
void	setEdgeHeadLeft (const double &edgeHeadLeft)	edgeHeadLeft	Setzt für die Größe des Pfeils (links) der Kante "edgeHeadLeft".
double	edgeHeadRight () const	edgeHeadRight	Gibt die Größe des Pfeils (rechts) der Kante zurück.

void	setEdgeHeadRight (const double &edgeHeadRight)	edgeHeadRight	Setzt für die Größe des Pfeils (rechts) der Kante "edgeHeadRight".
double	fluxValue () const	fluxValue	Gibt den Fluss-Wert der Kante zurück.
void	setFluxValue (const double &fluxValue)	fluxValue	Setzt für den Fluss-Wert der Kante "fluxValue".
QString	fluxDirectionString () const	fluxDirectionString	Gibt die Fluss Richtung der Kante in Form eines Strings zurück. Dieser kann "forward" oder "reverse" sein
void	setFluxDirectionString (const QString &fluxDirection)	fluxDirectionString	Setzt für die Fluss-Richtung der Kante "fluxDirectionString".
FluxDirection	fluxDirection () const	fluxDirection	Gibt den Enumerator-Wert "FluxDirection " der Kante zurück.
void	setFluxDirection (const FluxDirection &fluxDirection)	fluxDirection	Setzt für den Enumerator-Wert "FluxDirection " der Kante "fluxDirection".
EdgeLineType	edgeLineType () const	edgeLineType	Gibt den Enumerator-Wert "EdgeLineType" der Kante zurück.
void	setEdgeLineType (const EdgeObject::EdgeLineType &edgeLineType)	edgeLineType	Setzt für den Enumerator-Wert "EdgeLineType" der Kante "edgeLineType".
EdgeDirection	edgeDirection () const	edgeDirection	Gibt den Enumerator-Wert "EdgeDirection " der Kante zurück.
void	setEdgeDirection (const EdgeObject::EdgeDirection &edgeDirection)	edgeDirection	Setzt für den Enumerator-Wert "EdgeDirection "der Kante "edgeDirection".
PaintType	paintType () const	paintType	Gibt den Enumerator-Wert "PaintType" der Kante zurück.
void	setPaintType (const PaintType &paintType)	paintType	Setzt für den Enumerator-Wert "PaintType"der Kante "paintType".
QPointF	startPoint () const	startPoint	Gibt den Start Punkt der Kante zurück.
QPointF	endPoint () const	endPoint	Gibt den End Punkt der Kante zurück.
QVector< QPointF >	edgeHandleList () const		Gibt einen Vektor mit 2D Punkten der Kanten Ziehpunkte zurück die der Kante zugeordnet sind.
void	addEdgeHandle (const QPointF &edgeHandle)		Fügt zu dem Vektor der Kanten Ziehpunkte den Kanten Ziehpunkt "edgeHandle" zu.
void	switchStartEndNode ()		Wechselt Start- und Endknoten.
void	updatePosition ()		Aktualisiert die Position der Kante: Passt hierbei den Startpunkt und Endpunkt den Positionen der zugewiesenen Start- und Endknoten an.

QLineF	line () const	line	Gibt die Linie der Kante zurück.
bool	contains (const QPointF &point) const		Gibt zurück, ob die Kante den Punkt "point" schneidet.

Klasse *GroupObject*

Tabelle A.5: Übersicht der Methoden von *GroupObject* und die scriptbaren Funktionen.

Rüchgabe	Methode	Attribut (*QProperty*)	Beschreibung
int	groupId () const	groupId	Gibt die Gruppen-ID zurück.
void	setGroupId (const int &groupId)	groupId	Setzt für die Gruppen-ID "groupID".
QString	groupName () const	groupName	Gibt den Gruppen-Namen zurück.
void	setGroupName (const QString &groupName)	groupName	Setzt für den Gruppen-Namen "groupName".
QPointF	pos ()	position	Gibt die Gruppen-Position innerhalb der Netzwerksszene zurück.
void	setPos (const QPointF &pos)	position	Setzt die Gruppen-Position innerhalb der Netzwerkszene auf die Position "pos".
virtual QString	getInformation ()	information	Gibt die Gruppen-Information zurück.
virtual bool	isSelected () const	selected	Gibt zurück, ob die Gruppe selektiert ist.
virtual void	setSelected (bool selected)	selected	Wenn "selected" wahr ist, wird die Gruppe selektiert, andernfalls nicht.
bool	paintLabel ()	paintLabel	Gibt zurück, ob das Label der Gruppe gerendert wird.
void	setPaintLabel (bool paintLabel)	paintLabel	Wenn "paintLabel" wahr ist, wird das Label der Gruppe gerendert, andernfalls nicht.
PaintType	paintType () const	paintType	Gibt den Enumeratorwert von PaintType zurück.
void	setPaintType (const PaintType &paintType)	paintType	Setzt für den Enumeratorwert von PaintType "paintType".
bool	closePath ()	closePath	Gibt zurück, ob die Gruppe geschlossen dargestellt wird ist. Dieses betrifft die Darstellung im "Weg-Modus"
void	setClosePath (bool closePath)	closePath	Wenn "closePath" wahr ist, wird Gruppe geschlossen dargestellt, andernfalls nicht.
QString	groupNameType () const	groupNameType	Gibt die Gruppen-Namen-Typen zurück.

void	setGroupNameType (const QString &groupNameType)	groupNameType	Setzt für den Gruppen-Namen-Typen "groupNameType".
GroupType	groupType () const	groupType	Gibt den Enumeratorwert von GroupType zurück.
void	setGroupType (const GroupObject::GroupType &groupType)	groupType	Setzt für den Enumeratorwert von GroupType "groupType".
void	updatePosition ()		Aktualisiert die Position der Gruppe, passt hierbei die Umrandung den Knoten an
QVector< QPointF >	polygonPoints () const	polygonPoints	Gibt einen Vektor mit 2D Punkten der Knoten zurück die der Gruppe zugeordnet sind.
QList< NodeObject * >	getNodes ()	getNodes	Gibt eine Liste mit Pointern auf die Knoten zurück die der Gruppe zugeordnet sind.
QList< EdgeObject * >	getEdges ()	getEdges	Gibt eine Liste mit Pointern auf die Kanten zurück die der Gruppe zugeordnet sind.
QList< QGraphicsItem * >	getGraphicsItems ()	graphicsItems	Gibt eine Liste mit Pointern auf die Knoten und Kanten als QGraphicsItem gecastet zurück die der Gruppe zugeordnet sind.
QList< EdgeObject * >	getOutgoingEdges ()	outgoingEdges	Gibt eine Liste mit Pointern auf die Kanten zurück die zu Knoten gehen, die nicht der Gruppe zugeordnet sind.

Klasse *GraphicsViewTransformWidget*

Tabelle A.6: Übersicht der Methoden von *GraphicsViewTransformWidget* und die skriptbaren Funktionen.

Rüchgabe	Methode	Beschreibung
QPointF	getCenterPoint ()	Gibt den momentanen Center-Punkts des Views zurück.
void	setCenterPoint (QPointF center)	Setzt für den momentanen Center-Punkts des Views "center".
qreal	offsetX () const	Gibt den letzten Wert der Verschiebung in x-Richtung zurück.
void	setOffsetX (const qreal &offset)	Verschiebt die Szene um den Wert "offset" in x-Richtung.
qreal	offsetY () const	Gibt den letzten Wert der Verschiebung in y-Richtung zurück.
void	setOffsetY (const qreal &offset)	Verschiebt die Szene um den Wert "offset" in y-Richtung.
void	zoomIn ()	Zoomt in die Szene herein.
void	zoomOut ()	Zoomt aus der Szene heraus.
void	rotateLeft ()	Rotiert die Scene nach links.
void	rotateRight ()	Rotiert die Szene nach rechts.
void	left ()	Verschiebt die Szene nach links.
void	right ()	Verschiebt die Szene nach rechts.
void	up ()	Verschiebt die Szene nach oben.
void	down ()	Verschiebt die Szene nach unten.
void	addOffsetPoint (QPointF offset)	Verschiebt die Szene um den Punkt "offset".
void	resetView ()	Setz die Transformationsmatrix zurück.

qreal	getZoom ()	Gibt den momentanen Zoomfaktor (0- 100%) des Views zurück.
void	setZoom (qreal scale)	Setzt für den Zoomfaktor (0- 100%) den Wert "scale".
void	fitSelected ()	Fokussiert automatisch auf die momentan selektierten Netzwerkobjekte, und passt den sichtbaren Ausschnitt des Views diesen an.
void	fitAll ()	Fokussiert automatisch auf alle Netzwerkobjekte, und passt den sichtbaren Ausschnitt des Views diesen an.
void	fitInView (AbstractNetworkContainer *container)	Fokussiert automatisch auf den NetworkContainer, und passt den sichtbaren Ausschnitt des Views den Knoten, Kanten und Gruppen des NetworkContainer an.
void	fitInViewItem (QGraphicsItem *item)	Fokussiert automatisch auf das übergebene QGraphicsItem, und passt den sichtbaren Ausschnitt des Views dem QGraphicsItem an.
void	fitInViewItemsRelativeBounded (QList< QGraphicsItem * > items)	Fokussiert automatisch auf das übergebene QGraphicsItem und passt den sichtbaren Ausschnitt des Views dem QGraphicsItem an. Hierbei wird etwas herausgezoomt.
void	fitInViewSelected ()	Fokussiert automatisch auf die momentan selektierten Netzwerkobjekte, und passt den sichtbaren Ausschnitt des Views diesen an.
void	fitInViewAll ()	Fokussiert automatisch auf alle Netzwerkobjekte, und passt den sichtbaren Ausschnitt des Views diesen an.
void	fitInView (const QRectF &rect)	Fokussiert automatisch auf das übergebene Rechteck, und passt den sichtbaren Ausschnitt des View diesem an.
void	fitInViewItems (QList< QGraphicsItem * > items)	Fokussiert automatisch auf die übergebene QGraphicsItems, und passt den sichtbaren Ausschnitt des Views dieses an.
void	fitInViewItemsWithZoom (QList< QGraphicsItem * > items, qreal zoom)	Fokussiert automatisch auf die übergebenen QGraphicsItems, und passt den sichtbaren Ausschnitt des Views dieses an, wobei ein zusätzlicher Zoomfaktor angegeben wird.
void	fitInViewItemsWithRelativeZoom (QList< QGraphicsItem * > items, qreal zoom)	Fokussiert automatisch auf die übergebenen QGraphicsItems, und passt den sichtbaren Ausschnitt des Views dieses an, wobei etwas herausgezoomt wird.
void	moveToPoint (const QPointF &newPosition)	Fokussiert automatisch auf die übergebene Position.
void	moveToPointAnimated (const QPointF &newPosition)	Fokussiert automatisch auf die übergebene Position. Dieses geschieht animiert.
QPointF	getPoint ()	Gibt den momentanen Mittelpunkt der Szene bezüglich des momentanen Bildausschnitts zurück.
void	setScale (qreal scale)	Setzt für den Zoomfaktor (0- 100%) den Wert "scale".
void	resetMatrix ()	Setzt die Transformationsmatrix zurück.

Klasse *NetworkDocument*

Tabelle A.7: Übersicht der Methoden von *NetworkDocument* und die skriptbaren Funktionen.

Rückgabe	Methode	Beschreibung
void	layoutPainterPath (QString name)	Layoutet die selektierten Knoten dem nach dem Weg des übergebenem Strings.
void	setRainbow ()	Färbt die selektierten Knoten in Regenbogenfarben.
NodeObject *	getNodeByScriptObjectName (QString name)	Liefert den Pointer auf einen Knoten für den ScriptObjectName wenn dieser vorhanden ist, sonst 0.
EdgeObject *	getEdgeByScriptObjectName (QString name)	Liefert den Pointer auf eine Kante für den ScriptObjectName wenn dieser vorhanden ist, sonst 0.
QString	getRandomNodeName ()	Gibt einen zufälligen Knoten-Namen zurück.
void	cutSelection ()	Entfernt die selektierten Netzwerkobjekte aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	copySelection ()	Kopiert die selektierten Netzwerkobjekte aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	pasteSelection ()	Fügt die kopierten Netzwerkobjekte in das aktuell ausgewählte Netzwerk (NetworkContainer) ein.
void	deleteSelection ()	Entfernt die selektierten Netzwerkobjekte aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	selectAll ()	Selektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	selectAllNodes ()	Selektiert sämtliche Knoten für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	selectAllEdges ()	Selektiert sämtliche Kanten für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	selectAllGroups ()	Selektiert sämtliche Gruppen für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	selectAllInView ()	Selektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	selectAllInViewNodes ()	Selektiert sämtliche Knoten für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	selectAllInViewEdges ()	Selektiert sämtliche Kanten für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.

void	selectAllInViewGroups ()	Selektiert sämtliche Gruppen für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	deselectAll ()	Deselektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	deselectAllNodes ()	Deselektiert sämtliche Knoten für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	deselectAllEdges ()	Deselektiert sämtliche Kanten für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	deselectAllGroups ()	Deselektiert sämtliche Gruppen für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	deselectAllInView ()	Deselektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	deselectAllInViewNodes ()	Deselektiert sämtliche Knoten für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	deselectAllInViewEdges ()	Deselektiert sämtliche Kanten für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	deselectAllInViewGroups ()	Deselektiert sämtliche Gruppen für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	expandSelection ()	Expandiert die selektierten Knoten entlang ihrer zugewiesenen Kanten.
void	decreaseSelection ()	Verringert die selektierten Knoten entlang ihrer zugewiesenen Kanten.
void	toggleAll ()	Selektiert sämtliche Netzwerkobjekte, die nicht selektiert waren und deselektiert sämtliche Netzwerkobjekte, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer).
void	toggleAllNodes ()	Selektiert sämtliche Knoten, die nicht selektiert waren und deselektiert sämtliche Knoten, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) .
void	toggleAllEdges ()	Selektiert sämtliche Kanten, die nicht selektiert waren und deselektiert sämtliche Kanten, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) .
void	toggleAllGroups ()	Selektiert sämtliche Gruppen, die nicht selektiert waren und deselektiert sämtliche Gruppen. die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) .
void	toggleAllInView ()	Selektiert sämtliche Netzwerkobjekte, die nicht selektiert waren und deselektiert sämtliche Netzwerkobjekte, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	toggleAllInViewNodes ()	Selektiert sämtliche Knoten, die nicht selektiert waren und deselektiert sämtliche Knoten, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	toggleAllInViewEdges ()	Selektiert sämtliche Kanten, die nicht selektiert waren und deselektiert sämtliche Kanten, die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem momentanen Bildausschnitt.
void	toggleAllInViewGroups ()	Selektiert sämtliche Gruppen die nicht selektiert waren und deselektiert sämtliche Gruppen. die selektiert waren für das aktuell ausgewählte Netzwerk (NetworkContainer) und dem

		momentanen Bildausschnitt.
void	updateEdges (QList< EdgeObject * > edgeList)	Aktualisiert die Position der Kante aus der Liste, passt hierbei den Startpunkt und Endpunk den zugewiesenen Start und Endpunkt der Knoten an.
void	updateCurrentGroups ()	Aktualisiert die Position der Gruppen, passt hierbei die Umrandung den Knoten an.
void	updateGroups (QList< GroupObject * > groupList)	Aktualisiert die Position der Gruppen aus der Liste, passt hierbei die Umrandung den Knoten an.
void	addNode ()	Fügt einen neuen Knoten dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addNode (const QString &label)	Fügt einen neuen Knoten mit dem Label "label" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addNode (const QPointF &position)	Fügt einen neuen Knoten mit der Position "position" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addNode (const QString &label, const QString &scriptObjectName)	Fügt einen neuen Knoten mit dem Label "label" und dem Script-Objekt-Namen "scriptObjectName" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addEdge (NodeObject *startItem, NodeObject *endItem)	Fügt eine neue Kante mit den Startknoten "startItem" und Endknoten "endItem" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addEdge (EdgeObject *startItem, NodeObject *endItem)	Fügt eine neue Kante mit neu erzeugtem Startknoten und Endknoten "endItem" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu. Der neue Knoten wird in der Mitte der Kante "startItem" platziert.
void	addEdge (NodeObject *startItem, EdgeObject *endItem)	Fügt eine neue Kante mit den Startknoten "startItem" mit neu erzeugtem Endknoten dem aktuell ausgewählten Netzwerk (NetworkContainer) zu. Der neue Knoten wird in der Mitte der Kante "endItem" platziert.
void	addEdge (EdgeObject *startItem, EdgeObject *endItem)	Fügt eine neue Kante mit neu erzeugtem Startknoten und mit neu erzeugtem Endknoten dem aktuell ausgewählten Netzwerk (NetworkContainer) zu. Der neue Startknoten wird in der Mitte der Kante "startItem" platziert. Der neue Endknoten wird in der Mitte der Kante "endItem" platziert.
void	addEdge (const QString &label, const QString &scriptObjectName, QString nameNodeStart, QString nameNodeEnd)	Fügt eine neue Kante mit dem Label "label" und dem Script-Objekt-Namen "scriptObjectName" dem aktuell ausgewählten Netzwerk (NetworkContainer) zu. Der Startknoten und Endknoten wird hierbei nach den Strings "nameNodeStart" und "nameNodeEnd" gesucht.
void	addGroup (QList< NodeObject * > nodeList, QList< EdgeObject * > edgeList, QList< GroupObject * > groupList)	Fügt eine neue Gruppe mit den übergebenen Knoten, Kanten und Gruppen dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addNetworkContainer (QList< NodeObject * > nodeList, QList< EdgeObject * > edgeList, QList< GroupObject * > groupList)	Fügt ein neues Teilnetzwerk mit den übergebenen Knoten, Kanten und Gruppen dem NetworkObject zu.
void	addCommandBasedNode (NodeObject *node)	Fügt einen erstellten Knoten Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addCommandBasedEdge (EdgeObject *edge)	Fügt eine erstellte Kante Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addCommandBasedGroup (GroupObject *group)	Fügt eine erstellte Gruppe Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.

void	removeCommandBasedNode (NodeObject *node)	Entfernt einen erstellten Knoten Undo-basiert aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	removeCommandBasedEdge (EdgeObject *edge)	Entfernt eine erstellte Kante Undo-basiert aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	removeCommandBasedGroup (GroupObject *group)	Entfernt eine erstellte Gruppe Undo-basiert aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	addCommandBasedNodes (QList< NodeObject * > nodeList)	Fügt die erstellten Knoten in der Liste Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addCommandBasedEdges (QList< EdgeObject * > edgeList)	Fügt die erstellten Kanten in der Liste Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	addCommandBasedGroups (QList< GroupObject * > groupList)	Fügt die erstellten Gruppen in der Liste Undo-basiert dem aktuell ausgewählten Netzwerk (NetworkContainer) zu.
void	removeCommandBasedNodes (QList< NodeObject * > nodeList)	Entfernt die erstellten Knoten in der Liste Undo-basiert aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	removeCommandBasedEdges (QList< EdgeObject * > edgeList)	Entfernt die erstellten Kanten in der Liste Undo-basiert aus dem aktuell ausgewählten Netzwerk (NetworkContainer).
void	removeCommandBasedGroups (QList< GroupObject * > groupList)	Entfernt die erstellten Gruppen in der Liste Undo-basiert aus dem aktuell ausgewählte Netzwerk (NetworkContainer).
void	addCommandBasedNetworkContainer (AbstractNetworkContainer *networkContainer)	Fügt den erstellten NetworkContainer Undo-basiert dem Netzwerk (NetworkObject) zu.
void	addCommandBasedNetworkContainerList (QList< AbstractNetworkContainer * > networkContainerList)	Fügt den erstellten NetworkContainer in der Liste Undo-basiert dem Netzwerk (NetworkObject) zu.
void	removeCommandBasedNetworkContainer (AbstractNetworkContainer *container)	Entfernt den erstellten NetworkContainer Undo-basiert aus dem Netzwerk (NetworkObject).
void	removeCommandBasedNetworkContainerList (QList< AbstractNetworkContainer * > networkContainerList)	Entfernt den erstellten NetworkContainer in der Liste Undo-basiert aus dem Netzwerk (NetworkObject).
void	addCommandBasedNetworkContainerToNetworkContainer (AbstractNetworkContainer *networkContainer1, AbstractNetworkContainer *networkContainer2)	Fügt alle Knoten, Kanten und Gruppen aus dem NetworkContainer "networkContainer1" Undo-basiert dem NetworkContainer "networkContainer2" zu.
void	removeCommandBasedNetworkContainerFromNetworkContainer (AbstractNetworkContainer *networkContainer1, AbstractNetworkContainer *networkContainer2)	Entfernt alle Knoten, Kanten und Gruppen aus dem NetworkContainer "networkContainer1" Undo-basiert aus dem NetworkContainer "networkContainer2".
void	addCommandBasedNetworkObjectsToNetworkContainer (AbstractNetworkContainer *networkContainer, QList< NodeObject * > nodes, QList< EdgeObject * > edges, QList< GroupObject * > groups)	Fügt alle Knoten, Kanten und Gruppen aus den Listen Undo-basiert dem NetworkContainer "networkContainer" zu.
void	removeCommandBasedNetworkObjectsFromNetworkContainer (AbstractNetworkContainer *networkContainer, QList< NodeObject * > nodes, QList< EdgeObject * > edges, QList< GroupObject * > groups)	Entfernt alle Knoten, Kanten und Gruppen aus den Listen Undo-basiert aus dem NetworkContainer "networkContainer".
void	setCommandBasedNodesProperty (QList< NodeObject * > nodeList, QByteArray property, QVariant value)	Setzt für alle Knoten aus der Liste für das Attribut "property" Undo-basiert den übergebenen Wert "value".

void	addNodesToScriptEngine (QList< NodeObject * > nodeList)	Fügt die erstellten Knoten in der Liste der Skript-Engine zu. Hierbei erfolgt die Reistrierung durch den Script-Objekt-Namen.
void	addEdgesToScriptEngine (QList< EdgeObject * > edgeList)	Fügt die erstellten Kanten in der Liste der Skript-Engine zu. Hierbei erfolgt die Registrierung durch den Script-Objekt-Namen.
void	addGroupsToScriptEngine (QList< GroupObject * > groupList)	Fügt die erstellten Gruppen in der Liste der Skript-Engine zu. Hierbei erfolgt die Registrierung durch den Script-Objekt-Namen.
void	addNetworkObjectToScriptEngine (NetworkObject *networkObject)	Fügt alle Knoten, Kanten und Gruppen aus dem NetworkObject der Skript-Engine zu. Hierbei erfolgt die Registrierung durch den Script-Objekt-Namen.
void	addNetworkContainerToScriptEngine (AbstractNetworkContainer *networkContainer)	Fügt alle Knoten, Kanten und Gruppen aus dem NetworkContainer der Skript-Engine zu. Hierbei erfolgt die Registrierung durch den Script-Objekt-Namen.
QStringList	getLabelList (QString nameType)	Gibt eine Liste von Strings aller Knoten-Label des Netzwerk-Objekts zurück.
void	setNameToNavigate (QString name)	Fokussiert automatisch auf den ersten Knoten, welche den Knoten-Namen "name" besitzt.
void	setLabelNameToNavigate (QString labelName)	Fokussiert automatisch auf den ersten Knoten, welche das Label "labelName" besitzt.
QList< NodeObject * >	filterPathwayFrames (QList< NodeObject * > nodes)	Gibt eine Liste mit Pointern auf die Knoten aus der übergebenen Liste zurück, welche kein Rahmen (Stoffwechsel) darstellen.
static void	setRandomColor (QList< GroupObject * > groups)	Vergibt eine zufällige Farbe für die Gruppen aus der übergebenen Liste.
static void	setDimensionsToHalf (QList< NodeObject * > nodes)	Reduziert die Höhe und Breite der Knoten aus der übergebenen Liste um die Hälfte.
void	setSelected (QList< QGraphicsItem * > items)	Selektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer), die die übergebene Liste beinhaltet.
void	setNotSelected (QList< QGraphicsItem * > items)	Deselektiert sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer), die die übergebene Liste beinhaltet.
void	setHighlightSelected (QList< QGraphicsItem * > items)	Hebt sämtliche Netzwerkobjekte für das aktuell ausgewählte Netzwerk (NetworkContainer), die die übergebene Liste beinhaltet, hervor.

Inhalt der DVD-ROM

Tabelle A.8: Übersicht über den Inhalt der DVD-ROM

Ordner	Beschreibung
Dissertation/	beinhaltet die Dissertation als PDF-Version
Beispieldateien/BRIME	beinhaltet die verwendeten Beispieldateien für BRIME
Beispieldateien/MapOmnia	beinhaltet die verwendeten Beispieldateien für MapOmnia
Datenbanken/	beinhaltet die Datenbankstrukturen sämtlich verwendeter Datenbanken
Dokumentationen/MapNet	beinhaltet die Doxygen-Dokumentation von MapNet
Dokumentationen/MapOmnia	beinhaltet die Doxygen-Dokumentation von MapOmnia
Programme/	beinhaltet Links zu verwendeten Programmen für Ubuntu sowie Windows
Libraries Ubuntu/	beinhaltet verwendete Libraries für Ubuntu
Libraries Windows/	beinhaltet verwendete Libraries für Windows
Quellcode/BRIME	beinhaltet sämtliche Quelltextdateien für BRIME
Quellcode/MapNet	beinhaltet sämtliche Quelltextdateien für MapNet
Quellcode/MapOmnia	beinhaltet sämtliche Quelltextdateien für MapOmnia
MapNet kompiliert Ubuntu	beinhaltet eine kompilierte Version des MapNet-Projekts für Ubuntu
MapNet kompiliert Windows	beinhaltet eine kompilierte Version des MapNet-Projekts für Windows ohne MySQL-Unterstützung
MapOmnia kompiliert Ubuntu	beinhaltet eine kompilierte Version des MapNet-Projekts für Ubuntu
MapOmnia kompiliert Windows	beinhaltet eine kompilierte Version des MapNet-Projekts für Windows ohne MySQL-Unterstützung

Lebenslauf

Persönliche Daten:

Name:	Timo Lühr
Familienstand:	ledig
Nationalität:	deutsch
Geburtsdatum:	08.08.1979
Geburtsort:	Cuxhaven
Wohnort:	Kreuzkampstraße 1 38114 Braunschweig

Ausbildung:

09/2008 – 02/2014 **Doktorarbeit in Bioinformatik und Biochemie**
Technische Universität Braunschweig
Thema: „Metabolic Network Data Integration and Visualization Software"

10/2002 – 08/2008 **Diplomstudiengang Biotechnologie**
Technische Universität Braunschweig
Stipendiat der TU Braunschweig im Wintersemester 2007/2008 und Sommersemester 2008
Notenschnitt: 1,2
Abschluss: Diplom-Biotechnologe

09/2004 **Vordiplom in Biotechnologie**
Technische Universität Braunschweig
Notenschnitt: 1,7

10/2000 – 09/2001 **Studiengang „ integrierte Informatik"**
Universität Paderborn

08/1999 – 06/2000 **Zivildienst- Sozialer Dienst im Seniorenwohnsitz**
Senioren-Ruhesitz-Peening, Warstein

06/1999 **Allgemeine Hochschulreife**
Friedrich-Spee-Gymnasium, Rüthen
Leistungskurse: Mathematik und Sozialwissenschaften mit Schwerpunkt Ökonomie
Notenschnitt: 1,7

Auslandsaufenthalte:

08/2005 – 02/2006 **Auslandstudium (in englischer Sprache)**
James Cook University, Australia
Absolvierung der Kurse: „Cell Regulation“, „Biotechnology“ und „Introductory Infectious Diseases and Immunobiology“

Praktische und wissenschaftliche Erfahrung:

06/2012 – heute **yasc Informatik GmbH**
Anstellung als Systemanalytiker

09/2008 – 05/2012 **Bioinformatiker (Doktorand)**
Institut für Bioinformatik und Biochemie, TU Braunschweig
Softwareentwicklung (C++) sowie Webentwicklung (PHP, JavaScript), Datenbanken, Anwendung von Graphentheorie und bioinformatischen Algorithmen, Hochschullehre

11/2007 – 08/2008 **Diplomarbeit**
Institut für Bioinformatik und Biochemie, TU Braunschweig
Titel „Hochdurchsatzmetabolomanalyse von *Corynebacterium glutamicum* Transposonmutanten und Entwicklung von statistischen Datenauswertungswerkzeugen“

09/2005 – 01/2006 **Studienarbeit**
Institut für Bioverfahrenstechnik, TU Braunschweig
Ziel war die Etablierung eines Selektionssystems auf adhärente Zellen von *Sphingomonas pituitosa* im Chemostaten

09/2001 – 05/2002 **Infineon Technologies/ Eupec GmbH, Warstein**
Anstellung als Operator für die qualitative Überprüfung von Thyristoren und Dioden

05/2001 – 06/2001 **Infineon Technologies/ Eupec GmbH, Warstein**
Anstellung als Operator für die qualitative Überprüfung von Thyristoren und Dioden

07/2000 – 09/2000 **Infineon Technologies/ Eupec GmbH, Warstein**
Anstellung als Operator für die qualitative Überprüfung von Thyristoren und Dioden

Studienbegleitende Tätigkeiten und Engagement:

05/2005 –	06/2006	**Studentische Hilfskraft in Forschung und Lehre** Institut für Bioverfahrenstechnik, TU Braunschweig Anstellung im Rahmen des Forschungsprojekts „Mikrobielle Produktion eines Exopolysaccharids durch *Sphingomonas pituitosa*“
10/2004 –	02/2005	**Studentische Hilfskraft in Forschung und Lehre** Institut für Physik der Kondensierten Materie, TU Braunschweig Betreuung eines physikalischen Praktikums für Biotechnologen

Kenntnisse und Freizeitinteressen:

Sprachkenntnisse	Englisch fließend
EDV-Kenntnisse	Microsoft Office & Open/ Libre Office, Datenbanken (SQL), Linux, verschiedene Programmiersprachen z.B. C++, Visual Basic, Perl & PHP
Führerschein	Klasse 3 (PKW)
Hobbies	Badminton, Volleyball, Fitness, Kochen, Musik, 3D-Visualisierungen und Animationen, wissenschaftliche Gesprächsrunden (monatlich)